AR-008-933

DSTO-TR-0067

An Evaluation of the Gear Averaging Signal Processor (GASP)

D.M. Blunt

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

An Evaluation of the Gear Averaging Signal Processor (GASP)

D.M. Blunt

Airframes and Engines Division
Aeronautical and Maritime Research Laboratory

DSTO-TR-0067

ABSTRACT

The Gear Averaging Signal Processor (GASP) is an IBM PC XT board designed and built by AMRL to calculate synchronous vibration signal averages in real-time. This report describes GASP at an initial stage of development in which it produces signal averages in a non-real-time mode. To evaluate GASP in this mode of operation, a comparison was made with a PC-based signal averaging system. This comparison shows that GASP works effectively, but is limited by the memory available on the board and the precision of some of its floating point calculations.

Approved for Public Release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Aeronautical and Maritime Research Laboratory
GPO Box 4331
Melbourne Victoria 3001 Australia

Telephone: (03) 626 7000
Fax: (03) 626 7999

AR No. 008-933
SEPTEMBER 1994

APPROVED FOR PUBLIC RELEASE

An Evaluation of the Gear Averaging Signal Processor (GASP)

EXECUTIVE SUMMARY

Synchronous signal averaging has emerged as a particularly useful technique for the analysis of vibration signals from multi-shaft gearboxes. This is because the technique effectively isolates the vibration from a particular gear shaft from all the other non-synchronous shafts (ie shafts rotating at different speeds), thus allowing each shaft to be individually examined for faults.

AMRL vibration analysis software currently computes synchronous signal averages on a desktop PC equipped with two plug-in expansion boards: an anti-aliasing filter board, and an analogue-to-digital converter board. The vibration signal is filtered and then digitised, together with a tachometer signal, by these two boards, and the data is then post-processed by the PC to compute the synchronous signal average.

In 1988 a proposal was put forward at AMRL to design and build a prototype processor which could compute synchronous vibration signal averages in real-time on a single plug-in board. The Instrumentation and Trials Group (ITG) at AMRL subsequently designed and built an IBM PC XT board to do this called the Gear Averaging Signal Processor (GASP). This report describes an initial version of GASP that was made available for testing. This version does not operate in real-time but instead uses a method of synchronous signal averaging similar to that performed on a desktop PC. This version of GASP was evaluated to determine whether it would work effectively in this mode of operation and to ascertain its limitations.

The evaluation has shown that the synchronous signal averages computed by GASP closely match those computed on a desktop PC, but that GASP is limited by its available memory and the precision of some of its floating point calculations. These factors limit the ability of GASP to compute signal averages of more than approximately 100 revolutions of the shaft-of-interest, when in practice it is usual to use more than 400 revolutions.

Author

D.M. Blunt

Airframes and Engines Division

Mr Blunt graduated from the University of Western Australia in 1989 with a Bachelor of Engineering (Mechanical) degree with first class honours. He commenced employment with the Aeronautical Research Laboratory in 1990 and spent two years on the engineer rotation scheme. Since 1992 Mr Blunt has been working in the field of gearbox fault detection using vibration analysis. During this time he has been involved with the development of vibration analysis technology for various Australian Defence Force aircraft gearboxes. These gearboxes include the main transmission of the Black Hawk and Seahawk helicopters, and the reduction gearbox of the Allison T56 turboprop found in the P3C Orion and C130 Hercules.

Contents

1. Introduction

Obtaining synchronous vibration signal averages from gearboxes has emerged as a very useful technique for providing early warning of gear defects. This is because the averaging process can isolate the vibration from a particular gear shaft and thus enhance the diagnosis of faults on that shaft [Refs. 1,2,3].

A synchronous vibration signal average for a particular shaft is obtained by sampling a vibration signal at a rate N times the shaft frequency, so that the sample record consists of groups of N samples per revolution of the shaft, and then ensemble averaging the groups of samples (ie. first sample of each group, second sample, etc.) over the required number of revolutions of the shaft. Synchronizing the sampling with the shaft speed requires a gearbox speed reference signal, which is usually derived from a tachometer pulse or AC electrical generator directly driven by the gearbox. The speed of each shaft in the gearbox can then be related back to the frequency of this signal through the gear ratios.

In 1988 a proposal was put forward at AMRL to design and build a prototype processor which could compute synchronous vibration signal averages in real-time. The Instrumentation and Trials Group at AMRL subsequently designed and built a IBM PC XT board to do this called the Gear Averaging Signal Processor (GASP). The version of GASP described in this report does not in fact operate in real time, but first acquires the data and then post-processes it in a manner similar to the AMRL PC-based system, which is used for helicopter transmission fault detection.

A comparison of GASP with the PC-based system, and an evaluation of the numerical precision of GASP, has been made to determine whether GASP will work satisfactorily in this non-real-time mode of operation.

2. Description of GASP

GASP has eight analogue vibration channels and one tachometer channel. A functional diagram is shown in Figure 1.

The vibration channels are routed through a combined multiplexer and variable gain amplifier, though a low pass anti-aliasing filter with a variable corner frequency, to a 16-bit analogue-to-digital converter (ADC). The ADC is controlled by a digital signal processor (DSP) which can be programmed to sample the amplified and filtered signal at various rates and store the data into the 512 kB of high speed random access memory (RAM) on the board. Note that in this evaluation only 256 kB of RAM was installed, as this was all that was available at the time.

The tachometer signal is fed into a comparator that, in combination with an up-counter and latch, and a 25 MHz clock, counts the number of clock pulses between positive transitions of the signal past a reference level (which can be

adjusted to suit the signal). Each transition causes the current value of the up-counter to be latched and read by the DSP, while simultaneously resetting the counter to zero. The DSP then stores this signal period measurement into the on-board RAM.

A signal average is computed in two parts: data acquisition, and signal averaging.

Data is acquired by down-loading a data acquisition program from the host PC into the DSP and executing it. The data acquisition program instructs the DSP to sample the vibration signal at a fixed rate, and simultaneously measure the tachometer signal periods, for a set length of time. Both sets of data are stored in the on-board RAM for the signal averaging program to use.

On completion of the data acquisition, the signal averaging program is down-loaded into the DSP and executed. This program uses the known ratio of the shaft frequency to the tachometer signal frequency, the tachometer period data, and a cubic interpolation algorithm [Ref. 4] to digitally re-sample the vibration data at a rate N times the shaft frequency, where N is any desired integer. Groups of N re-sampled data points are then ensemble averaged over the required number of revolutions of the shaft and the resulting signal average is transferred to the PC memory.

3. Description of the PC-Based System

The PC-based system uses two commercially available PC boards. The first is an Onsite Instruments Techfilter board with programmable low-pass filters for 16 differential channels, and the second is a Data Translation DT2821-G-8DI board incorporating a 12 bit analogue-to-digital converter with 8 differential channels. The first 8 Techfilter board channel outputs are connected to the DT2821-G-8DI board analogue channel inputs. All signals, including the tachometer signal, pass through the Techfilter board to the DT2821-G-8DI board, although only the vibration signals are filtered. Both boards have the facility to amplify the signals, although in practice only the amplifier on the DT2821-G-8DI board is used.

Computation of a signal average is performed in three parts: data acquisition, intermediate processing, and signal averaging.

Data is acquired by executing the data acquisition program. This samples both the vibration and tachometer signals simultaneously at identical rates for a set length of time, and transfers the data to a RAM disk file.

The intermediate processing involves two programs. The first converts the vibration data to engineering units, taking into account the transducer sensitivity and signal amplification, and the second extracts the tachometer signal period information. The latter is derived by counting the number of whole and partial sample periods between the negative-to-positive zero-crossings of the

tachometer signal. The zero-crossings are determined through linear interpolation of the pairs of samples points occurring on either side of 0 Volts.

Finally, the signal averaging program uses the ratio of the shaft frequency to the tachometer signal frequency, the tachometer period data, and a cubic interpolation algorithm to digitally re-sample the vibration data at a rate N times the shaft frequency, where N is any desired integer. Groups of N re-sampled data points are then ensemble averaged over the required number of revolutions of the shaft to obtain the signal average.

4. Comparison Tests

GASP and the PC-based system were compared by computing identical signal averages on each system and examining the differences between the two sets of signal averages. The signal averages were identical with respect to: sampling frequency, filter frequency, sample points per shaft revolution, and shaft revolutions averaged.

Two vibration recordings were used for this comparison. Both were made on Brüel & Kjær Model 7003 FM tape recorders from accelerometers mounted on:

a) an AMRL spur gear rig (a simple two shaft gearbox), and

b) a RAN Sea King helicopter main rotor gearbox (a complex multi-shaft gearbox).

The signal averages computed are summarized below and in Table 1. Note that the number of shaft revolutions over which the signals were averaged was limited by the memory available on GASP (256 kB). More shaft revolutions (approximately double) could have been averaged if GASP had been fully equipped with 512 kB, but this would still have fallen well short of the usual number performed on the PC-based system; normally in the order of several hundred shaft revolutions.

All signal averages computed consisted of 2048 sample points per shaft revolution. Fewer points could have been used for some signal averages, but 2048 points reduced the computation time for the COMPSIG program (see section 4.3) to find the phase difference between the signal averages.

Note that the conversion of the vibration signals from Volts to 'G' was not performed as this was incidental to the analysis.

4.1. AMRL Spur Gear Rig Signal Averages

The AMRL spur gear rig consists of just two gears; an input pinion with 27 teeth, and an output gear with 49 teeth. Signal averages were computed for both

gears. The tachometer signal on this tape consists of one square pulse per revolution of the output gear, with a frequency of ~21.5 Hz.

4.2. Sea King Signal Averages

The RAN Sea King tape 4/88 was used. Signal averages were computed for the planet carrier, the crown-wheel shaft, and the input pinion shaft. The tachometer signal on this tape is the AC generator signal with a frequency of ~ 400 Hz.

Table 1: GASP and PC-System Signal Average Details

Signal Average	Teeth	Gear Ratio	Sample Freq. (Hz)	Filter Freq. (Hz)	Points	Rev's Averaged
Gear Rig Input	27	49/27	16666.7	6000	2048	245
Gear Rig Output	49	1/1	16666.7	6000	2048	135
S/K Planet Carrier	196	3024/357975	8000.0	2600	2048	50
S/K Crown-wheel	85, 129	560/14319	19230.8	6800	2048	100
S/K Input Gear	25, 109	1904/14319	45454.5	15000	2048	125

4.3. Signal Average Comparison Program – COMPSIG

COMPSIG (see Appendix A for code) was written to facilitate comparison of the signal averages calculated by each system by removing the following spurious differences.

a) DC offsets The mean value of each signal average was subtracted from each point in the signal average.

b) Minor filter attenuation differences near the cut-off frequency The signal averages were additionally low-pass filtered by calculating their FFTs, setting the frequency components above a (user selected) point just below the anti-aliasing filter cut-off to zero, and converting the FFTs back to the time domain.

c) Phase difference The phase difference was calculated by determining the maximum of the cross-correlation function of the two signal averages, and removed by multiplying one signal average by a time-shift vector.

d) Minor signal amplification differences The signal averages were normalized to an RMS value of 1.

4.4. Comparison Procedure

Two comparisons were made between the two sets of signal averages so as to highlight the anti-aliasing filter differences. In the first comparison, COMPSIG was used to remove all the spurious differences listed in section 4.3 except the filter differences, and in the second comparison COMPSIG was used to remove all the spurious differences.

The results of each comparison have been combined into a single figure, in which:

i) part (a) shows a plot of the COMPSIG-processed PC signal average (the RMS value of this plot is always 1.00), and

ii) part (b) shows the difference between part (a) and the COMPSIG-processed GASP signal average. When multiplied by 100%, the RMS value of this plot can be interpreted as a percentage residual error.

4.5. Examples

To be better able to interpret the degree of agreement between the signal averages in the following discussion, a sample signal average (the spur gear rig input shaft) has been compared with two phase-shifted versions of itself. The first is phase-shifted by 0.5 sample points (0.5 x 360/2048 = 0.09°), and the second is phase-shifted by 1 sample point (0.18°).

These comparisons can be seen in Figures 2 and 3. In these figures, part (a) shows the signal average after COMPSIG processing, and part (b) shows the difference between it and the phase-shifted version of itself. It can be seen that even with these very small phase shifts the RMS values are quite large at 0.0735 and 0.1467 respectively.

On the basis of these examples, a good match between signal averages could be said to occur if the RMS value falls below 0.070.

5. Comparison Test Results

Areas that may cause differences between the signal averages computed by each system are listed below.

a) ADC precision GASP has a 16 bit ADC which will provide a greater dynamic range than the 12 bit ADC in the PC-based system (90 dB and 66 dB respectively). However, the B&K 7003 recorder has a dynamic range of only 44 dB so the ADC precision will probably not affect the results greatly.

b) Different tachometer signal period measurement methods GASP should be more accurate as it measures the period in 25 MHz clock cycles regardless of the vibration signal sampling frequency, while the PC-based system samples the tachometer signal at a much slower rate (the same rate as the vibration signal) and linearly interpolates the zero-crossings from this data. It would be expected that inaccuracy in the period measurement would have a low pass filtering effect on the signal average as it causes the vibration data to be re-sampled with slight perturbations from the precise, shaft-synchronous, sampling positions.

c) Different anti-aliasing filter characteristics Slightly different ripples in the pass band, phase non-linearity, and behaviour in the vicinity of the corner frequency will affect the relative frequency content of the signal averages. The filter characteristics can be found in Appendix B.

d) Different processing precision The PC-based system performs all floating point processing with 64 bit numbers while GASP is restricted to 32 bit numbers. This is discussed in detail in section 6.

e) Noise susceptibility Investigations revealed that unless the input wires from GASP were twisted, a small (~0.01 Vrms) 50 Hz pulse would be induced by the electricity mains. This did not occur on the PC-based system, but the filters on both systems were found to introduce DC offsets when frequencies above approximately twenty times the corner frequency were present in the input signal.

5.1. Spur Gear Rig Tape

The two systems were expected to yield closely matching signal averages from the data generated by this simple gear configuration, since there is only one gear mesh in this gearbox and it is synchronous with both shafts. This will circumvent any difficulties that may be caused by the limited number of shaft revolutions over which the vibration is averaged.

5.1.1. Input Shaft

The signal averages and their spectra, as computed by both systems for the input gear shafts, are shown in Figures 4 and 5. While it is difficult to discern differences in the averages, examination of the spectra reveals a small difference in the relative magnitudes of the highest meshing harmonic and its sidebands. This is probably due to a combination of the different anti-aliasing filter roll-off rates near the corner frequency, inaccuracies in setting the corner frequency, and possible inaccuracies in the tachometer signal period measurement.

Figures 6 and 7 show the comparisons between the signal averages after they have been processed by COMPSIG: Figure 6 without the additional low-pass

filtering, and Figure 7 with the highest gear mesh harmonic and its sidebands removed (ie. frequencies above 150 shaft orders). It can be seen that the additional filtering results in a very good match of the signal averages as the residual error is reduced to 1.57%.

5.1.2. Output Shaft

The results for the output shaft are very similar to the input shaft and can be found in Figures 8 to 11. In this case removing the highest meshing harmonic and its sidebands (ie. frequencies above 270 shaft orders) produces a very good match between the signal averages, with a residual error of 1.56%.

5.2. Sea King Tape

The Sea King gearbox is more complex and contains many strong vibration sources that are not synchronous with the shafts examined here. Therefore it is to be expected that the limited number of shaft revolutions that can be averaged by GASP will not be enough to attenuate all the non-synchronous vibration from the signal averages, and will result in poorer matches.

5.2.1. Planet Carrier

The planet carrier vibration signal is the easiest to extract from the overall signal as the vibration was recorded from an accelerometer mounted near the ring gear housing. The major meshing frequency synchronous with this shaft is that of the ring gear which has 196 teeth.

Plots of the planet carrier signal averages and their spectra, as calculated by both systems, can be found in Figures 12 and 13. It can be seen that the signal averages clearly show the amplitude modulation caused by the five planet gears rotating past the accelerometer location. The spectra also show the attenuation of the gear mesh frequency and its harmonics and the asymmetrical sidebands that are characteristic of an epicyclic gear [Ref. 5].

A difference between the spectra in relative magnitudes of the highest gear mesh harmonic and its sidebands is present here in the same manner as in the gear rig spectra, and is probably again due to slight differences between the anti-aliasing filters, and/or inaccuracies in the tachometer signal period measurement. There also appears to be more "noise" in these signal averages, as evidenced by the appearance of small frequency lines appearing between the gear mesh harmonics. This "noise" probably consists of the non-synchronous vibration that is incompletely attenuated by the averaging process.

Figures 14 and 15 show the comparisons between the two signal averages after processing by COMPSIG: Figure 14 without additional filtering, and Figure 15 with the highest meshing harmonic and sidebands removed (ie. frequencies above 700 shaft orders). The additional filtering does not improve the match as much as in the case of the gear rig signal averages, which is probably due to the increased "noise" level. Overall though, the match between the signal averages is still relatively good with only a 6.20% residual error after the additional filtering, despite the limited number of revolutions over which the signal was

averaged. This can be attributed to the strength of the vibration signal from the epicyclic gear train reaching the transducer.

5.2.2. Crown-wheel Shaft

The crown-wheel shaft has three gears: two crown-wheel gears which have 129 and 85 teeth respectively, and the sun gear of the epicyclic gear train which has 54 teeth. Vibration from the sun gear, however, will be attenuated as it does not mesh synchronously with the crown-wheel shaft.

The calculated signal averages and their spectra can be found in Figures 16 and 17. It is immediately obvious from these plots that the signal strength is much lower compared to the planet carrier shaft: 0.0323 Vrms compared to 0.1977 Vrms (for the PC-based system). Indeed, the crown-wheel shaft vibration level appears to be quite close to the "noise" floor. Additionally, the spectra reveal that there are significant differences in the magnitudes of the major frequency components across the whole frequency range.

Figures 18 and 19 show the differences between the signal averages after processing by COMPSIG, both with and without additional low pass filtering, although in this case it was difficult to decide where to set the cut-off frequency. Four hundred shaft orders was chosen because it was ~95% of the selected analogue filter cut-off frequency (approximately the same proportion used for the previous signal averages). It can be seen that the difference between the signal averages is approximately of the same magnitude as the signal averages themselves, with a residual error of 70.84% in the filtered case, confirming that they do not match well.

The factors listed at the start of Section 5. are unlikely to be major causes for the poor match in this case in light of the good agreement in the previous signal averages. Rather, these results indicate that the lack of good agreement is most probably due to the low signal-to-noise ratio: that is, the vibration signal from this shaft is so low compared to the other non-synchronous vibration that a signal average over only 100 shaft revolutions is insufficient to effectively isolate it. It was not possible to increase the length of the average to confirm this due to the lack of memory on GASP.

5.2.3. Input Pinion Shaft

The input pinion shaft has two gears: the pinion itself, which has 25 teeth; and another gear, with 109 teeth.

Figures 20 and 21 show the signal averages and their spectra calculated for this shaft. The RMS amplitudes for these signal averages, at 0.0441 Vrms and 0.0417 Vrms, are slightly higher than those for the crown-wheel shaft, but are of the same order of magnitude as, and exhibit similar frequency component differences to, the crown-wheel shaft signal averages.

Plots of the comparisons between the signal averages after processing by COMPSIG, both with and without additional low pass filtering, are shown in Figures 22 and 23. Again, it was difficult to decide where to set the cut-off

frequency, and consequently the same approach was taken as for the crown-wheel shaft signal averages in arriving at 235 shaft orders. The matches, though, are clearly not good in either case, with a residual error of 79.53% in the filtered case.

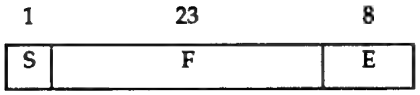
It is most likely that the lack of agreement between these signal averages is again due to the low signal-to-noise ratio of the vibration from this shaft, compared to the other non-synchronous vibration, and an insufficient number of shaft revolutions in the average.

6. GASP Precision

The DSP code used to calculate the signal averages for all the above tests can be found in Appendix C and is based around two nested loops. The outside loop steps through the number of shaft revolutions required, while the inside loop steps through the number of sample points per shaft revolution required. At each step of the inside loop the point index is divided by the number of points per shaft revolution, and the result added to the whole revolution index to get the number of revolutions so far. This floating point number is then multiplied by the gear ratio to give the corresponding number of tachometer signal periods. There follows a calculation in which this number is used to obtain a memory pointer which, if it were an integer, would point to the memory location where the corresponding vibration sample was stored. Since the result of the calculation is, in general, a floating point number, the memory pointer points to an intermediate memory location. A cubic interpolation procedure then uses the two sample values either side of the pointer to interpolate a value for this point. This value is then added to the sum of the previous values interpolated for this point from the prior steps of the outside loop. At the conclusion of the loops the summed values for each point are divided by the number of steps in the outside loop thus giving the signal average.

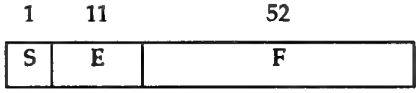
Ignoring the interpolation procedure, problems with the precision of the floating point calculations can occur at three places in the program: calculation of the number of revolutions of the shaft of interest, and the corresponding number of tachometer signal periods; calculation of the memory pointer; and summation of the interpolated sample values.

The floating point type used by the DSP program consists of 32 bits arranged in the following format. It has a 23 bit mantissa which allows for 7 significant digits in base 10.



$$N = \left[-2^{S} + 0.F\right] \times 2^{(E-128)}$$

By comparison, the double precision IEEE format used by the PC-based system consists of 64 bits arranged in the following format. It has a 52 bit mantissa which allows for 15 significant digits in base 10.



$$N = -1^{S} \times 1.F \times 2^{(E-1023)}, 0 < E < 2047$$

6.1. Calculation of the Number of Revolutions of the Shaft of Interest

This calculation has been described above. The problem lies with a variable of the DSP floating point type having enough significant digits to adequately represent the small increments that occur with each small step around the gear shaft of interest, when the number of whole revolutions gets large. The problem is carried over into the calculation of the number of tachometer signal periods, as it is found by multiplying this number by the gear ratio.

For N points around the shaft of interest there need to be enough decimal places to allow for increments of 1/N. For a signal average over several hundred shaft revolutions, however, there will only be 4 significant decimals places available (XXX.XXXX) once the number exceeds 100. This therefore limits the number of shaft revolutions that can be averaged to approximately 100 before inaccuracies develop. It would be expected that this problem would manifest itself as a worsening ability of GASP to resolve the higher frequencies as the number of shaft revolutions increases.

For the 2048 point signal averages of this evaluation the increment is exactly 0.00048828125, which requires at least 11 significant decimal places for there to be no accuracy problems. Note, however, that the signal average for the gear rig input shaft, which is 245 revolutions, does not appear to have suffered unduly from this problem. This may be attributable to removal of the higher frequencies by the anti-aliasing filter. The number of sample points per revolution of the shaft needed to resolve the filter cut-off frequency of 6000 Hz in this case is:

$$2 \times 6000 \frac{\text{points}}{\text{sec}} \div \left(21.5 \times \frac{49}{27} \frac{\text{rev}}{\text{sec}} \right) = 308 \frac{\text{points}}{\text{rev}}$$

This requires increments of 1/308 = 0.0032468, for which the 4 decimal places available do not incur as large an accuracy problem. The problem will become of greater significance, though, if the GASP memory were to be expanded to allow more shaft revolutions to be captured.

6.2. Calculation of the Memory Pointer

The memory pointer is initially set to point to a base word reference address representing the sample at the start of the second tachometer period; the first period is ignored. This address is not a real address as it has a fractional part representing the offset between the start of the period and the next value to be re-sampled. At each step of the inside loop the number of tachometer signal periods is split into integer and fractional parts. The integer part is used to increment the base word reference address to the sample at the start of each new period, and the fractional part is used to get a pointer to the sample address within the current period. Cumulative numerical error is avoided by storing the offset part of the base word reference address as an integer number of clock cycles.

The problem arises when the word reference address, which is a floating point number, reaches the upper limits of memory and leaves only a few significant digits available to represent the increments in the pointer. The maximum amount of memory available on GASP is 512 kB, or 262144 words, which needs 6 significant digits for the higher part of memory, leaving just 1 decimal place to represent the fractional part. Note that in this evaluation only 256 kB, or 131072 words, were used, and thus the problem is of slightly smaller magnitude in the signal averages computed here. Also note that the precision of the word reference address calculation is dependent on the precision of the tachometer signal period calculation. It would be expected that the problem would manifest itself as a worsening ability of GASP to resolve the higher frequencies as the word reference address increases.

Table 2 lists the increments in the word reference address pointer for the Sea King and gear rig signal averages. These figures have been calculated assuming tachometer signals of exactly 400 Hz and 21.5 Hz respectively. While there is no advantage in using more points per revolution of the shaft of interest in the signal average than there are samples, it is normal to use the next highest power of two to ease FFT calculations. It can be seen that more than one significant decimal place is needed in the pointer variable. The fact that the gear rig signal averages do not appear to have suffered unduly from this problem may again be attributable to the anti-aliasing filter removing the higher frequencies where the differences would arise. The problem will become of greater concern for longer signal averages which require more memory to store the data.

Table 2: Increments in the Memory Pointer

Shaft	Sample Freq	512 Pt Average	1024 Pt Average	2048 Pt Average	4096 Pt Average
Gear Rig Input	16666.7	0.8342743311	0.4171371656	0.2085685828	0.1042842914
Gear Rig Output	16666.7	1.514053416	0.7570267080	0.3785133540	0.1892566770
Planet Carrier	8000.0	4.624139695	2.312069848	1.156034924	0.5780174620
Crown-wheel	19230.8	2.400995609	1.200497805	0.6002489023	0.3001244512
Input Pinion	45454.5	1.669141333	0.8345706667	0.4172853333	0.2086426667

6.3. Summation of the Interpolated Sample Values

At each step of the inside loop the interpolated sample value for that point is added to the sum of the previous values for that particular point in the signal average. The problem can arise if the summation value becomes so large that the precision of the interpolated value is lost.

In the worst case the maximum value the summation will reach is the number of shaft revolutions in the signal average multiplied by the maximum value of the 16 bit ADC. With, say, a 500 revolution signal average this will be

$$500 \times 2^{15} = 1.638400 \times 10^{7}$$

The seventh significant figure in this case is in the 10's column meaning that everything less than this digit in the interpolated value would be lost. However, as the ADC is only precise down to the units column, and the worst case is very unlikely to happen, this should not cause problems.

7. Concluding Remarks

The relatively small memory capacity of GASP, which reflects its original design for real-time operation for which this would not be a handicap, severely limits the amount of data that it can capture. Compared to the PC-based system, where the memory capacity is only dependent on the PC, and may run into megabytes, this is a major limiting factor of GASP. It prevents averaging more than approximately 100 revolutions of the shaft of interest, when in practice it is usual to use between 400 to 1000.

Within this limitation, however, and others noted below, the favourable comparisons of the signal averages from the spur gear rig and the planet carrier of the Sea King main rotor gearbox have confirmed that GASP does operate correctly. They also indicate that the differences between the systems have not had a great effect on the results.

The lack of good agreement between the two sets of signal averages for the Sea King crown-wheel and input pinion shafts is most likely due to a combination of the low signal-to-noise ratios for the vibration signals from these shafts, and insufficient signal averaging. This could not be confirmed, however, due to the lack of memory on GASP preventing longer averaging of the signals.

Examination of the precision of the floating point calculations performed by GASP has revealed that the 32-bit floating point number type of the DSP is insufficient in certain circumstances. Specifically, in the calculation of the number of revolutions of the shaft of interest, and the calculation of the word reference pointer. This limits the number of shaft revolutions which can be averaged before numerical errors start to become significant to approximately 100.

Other information to come out of these tests includes:

a) Small DC offsets are introduced into the output of the anti-aliasing filters when there are frequency components above approximately 20 times the filter corner frequency.

b) The input leads to the GASP vibration input channels are very susceptible to picking-up interference and need to be twisted or otherwise shielded to minimize this.

8. Acknowledgements

The author acknowledges the assistance with the PC based signal averaging software given by B. Rebbechi and B.D. Forrester, and the assistance with the GASP hardware and software given by O.F. Holland, J.F. Harvey and I.M. Kerton.

9. References

1. Swansson, N.S. "Application of Vibration Signal Analysis Techniques to Condition Monitoring", Conference on Lubrication Friction and Wear in Engineering, Melbourne, 1980.

2. McFadden, P.D. "Advances in the Vibration Monitoring of Gears and Rolling Element Bearings", I.E. Aust/ R.Ae.S. Joint National Symposium, Melbourne, 1985.

3. McFadden, P.D. "Examination of a Technique for the Early Detection of Failure in Gears by Signal Processing the Time Domain Average of the Meshing Vibration", ARL Aero Propulsion Technical Memorandum 434, 1986.

4. McFadden, P.D. "Interpolation Techniques for the Time Domain Averaging of Vibration Data with Application to Helicopter Gearbox Monitoring", ARL Aero-Propulsion Technical Memorandum 437, 1986.

5. McFadden, P.D. and Smith, J.D. "An Explanation for the Asymmetry of the Modulation Sidebands about the Tooth Meshing Frequency in Epicyclic Gear Vibration", Proc. I. Mech. E., Part C, vol 199 (1985), pp 65-70.

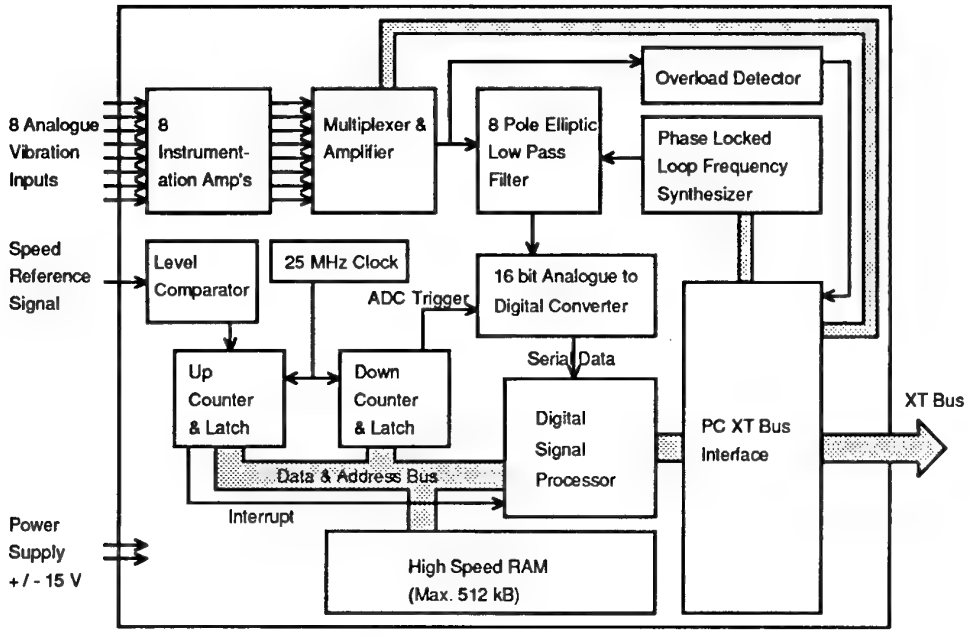


Figure 1: GASP Functional Diagram.

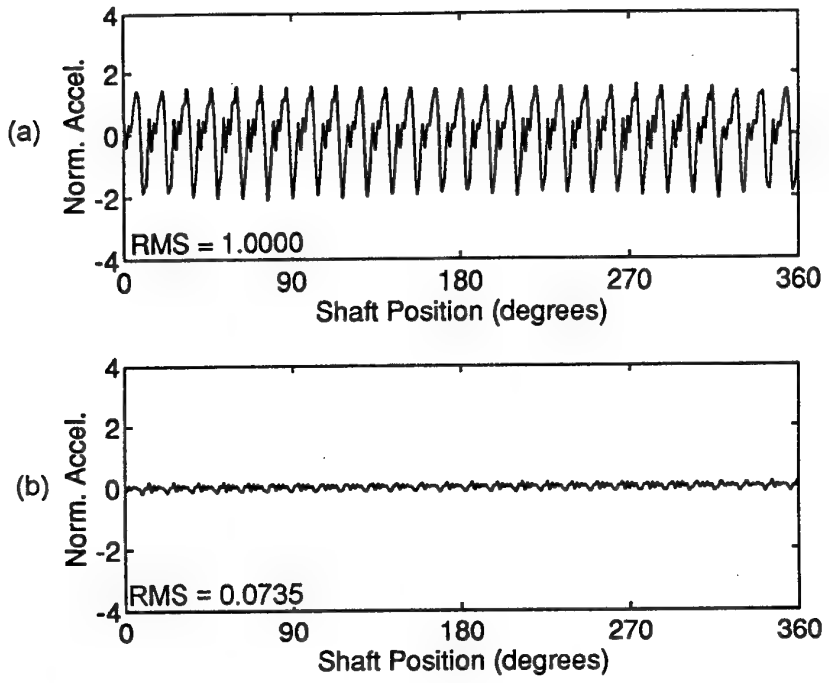


Figure 2: Comparison of a signal average with itself phase-shifted by 1/2 point (0.09°) (No additional filter) a) Signal average b) Difference

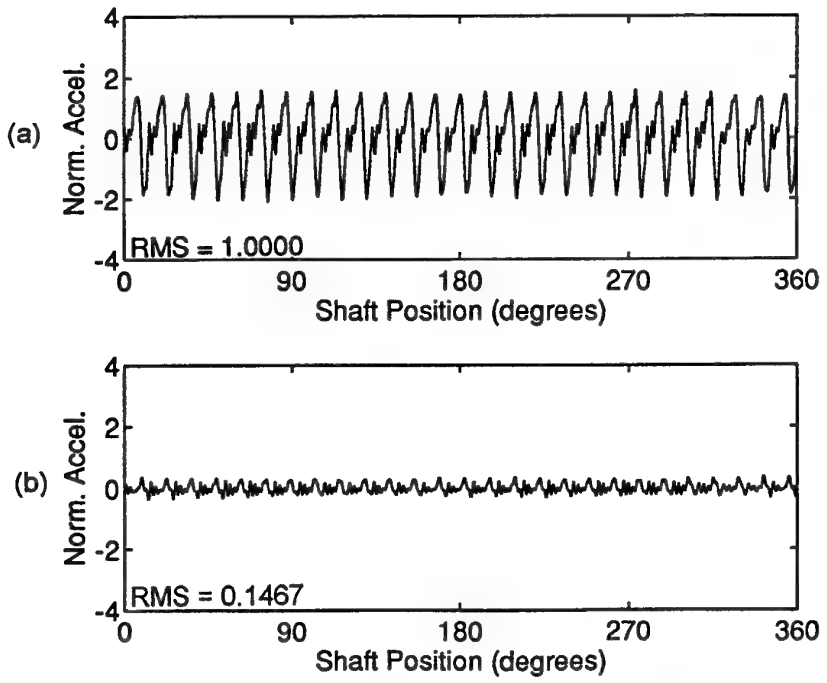


Figure 3: Comparision of a signal average with itself phase-shifted by 1 point (0.18°) (No additional filter) a) Signal average b) Difference

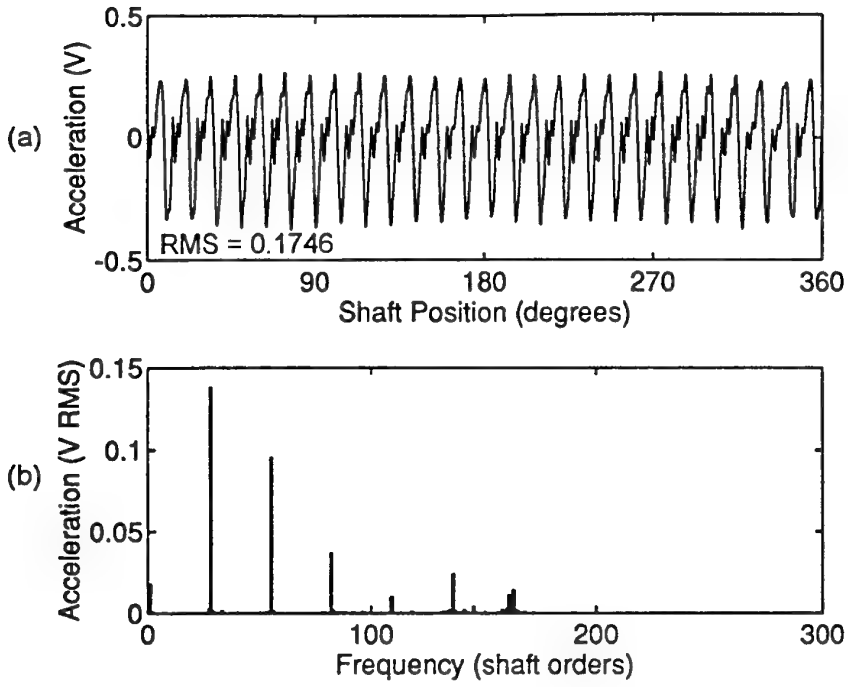


Figure 4: Gear rig input shaft – PC system
a) Signal average b) Spectrum

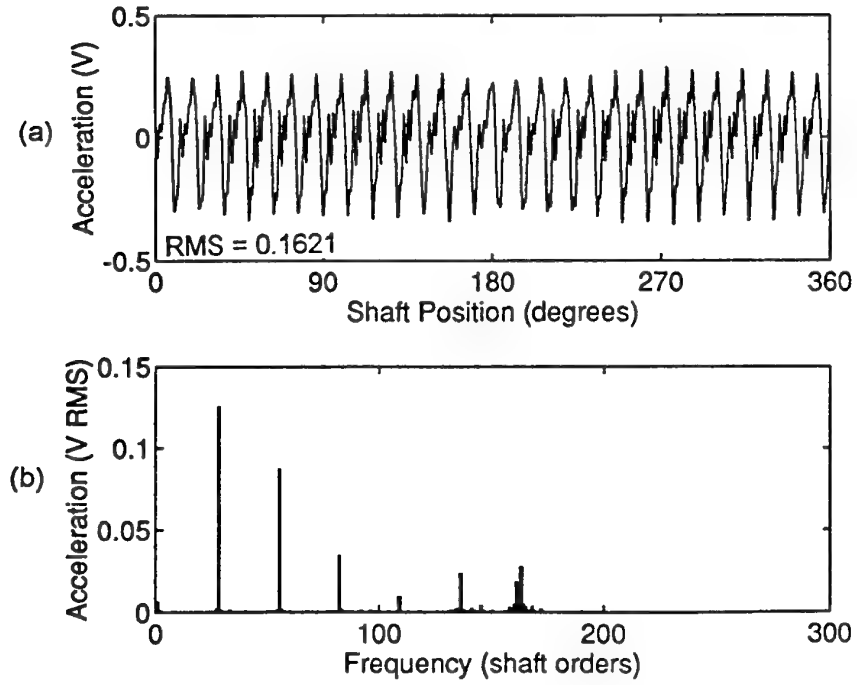


Figure 5: Gear rig input shaft – GASP
a) Signal average b) Spectrum

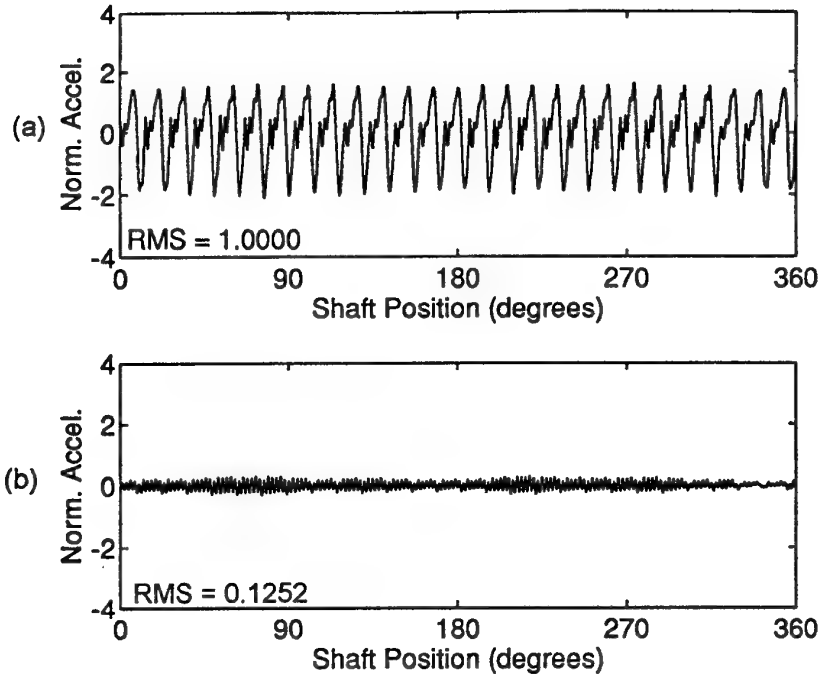


Figure 6: Comparison of normalized gear rig input shaft signal averages (No additional filter) a) Signal average b) GASP difference

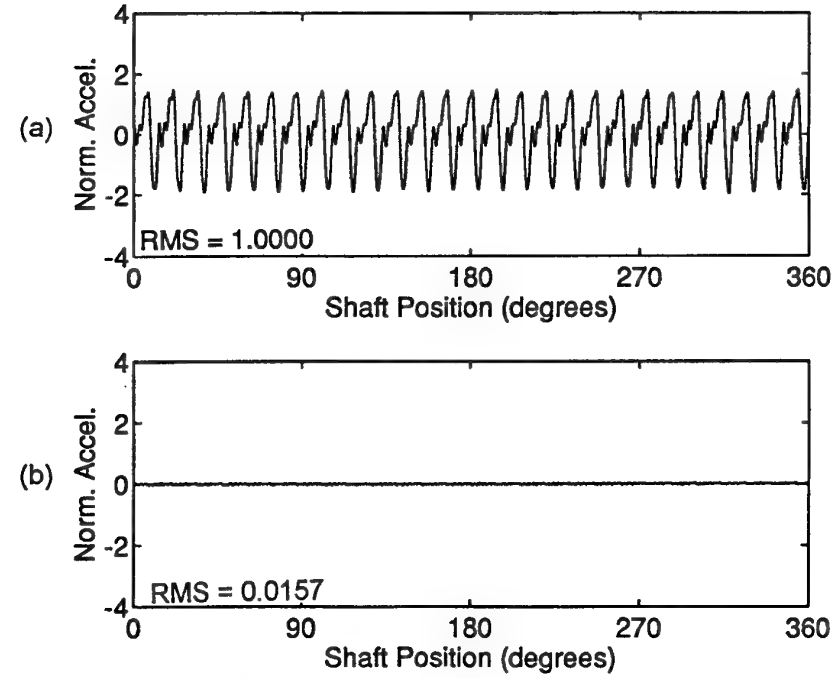


Figure 7: Comparison of normalized gear rig input shaft signal averages (Additional filter @ 150 orders) a) PC Signal average b) GASP difference

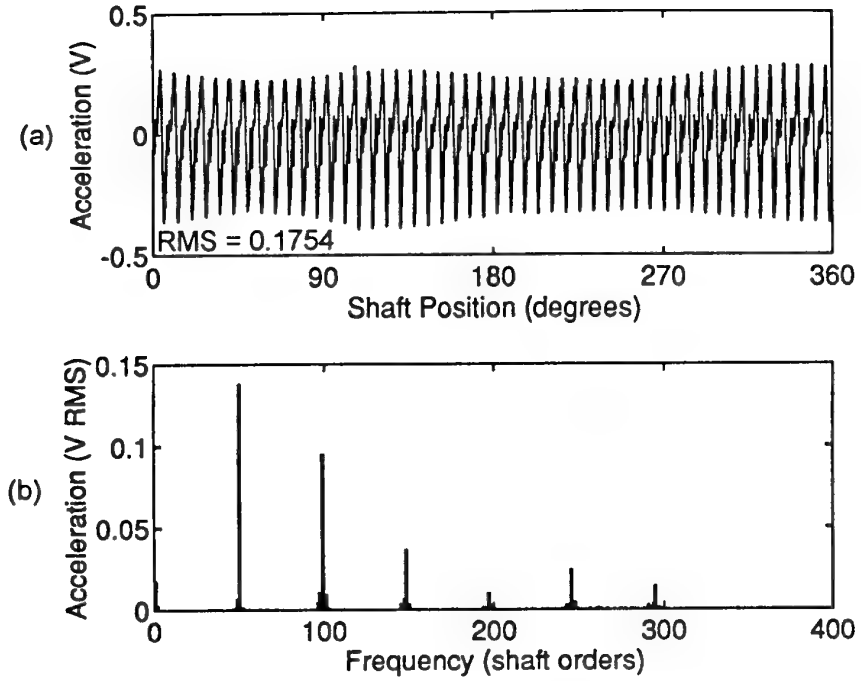


Figure 8: Gear rig output shaft – PC system
a) Signal average b) Spectrum

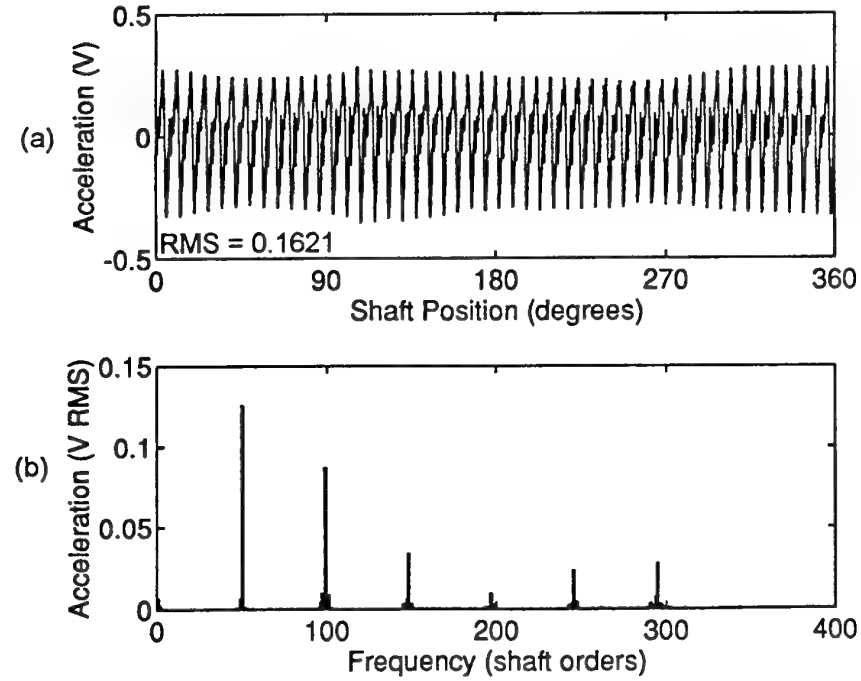


Figure 9: Gear rig output shaft – GASP
a) Signal average b) Spectrum

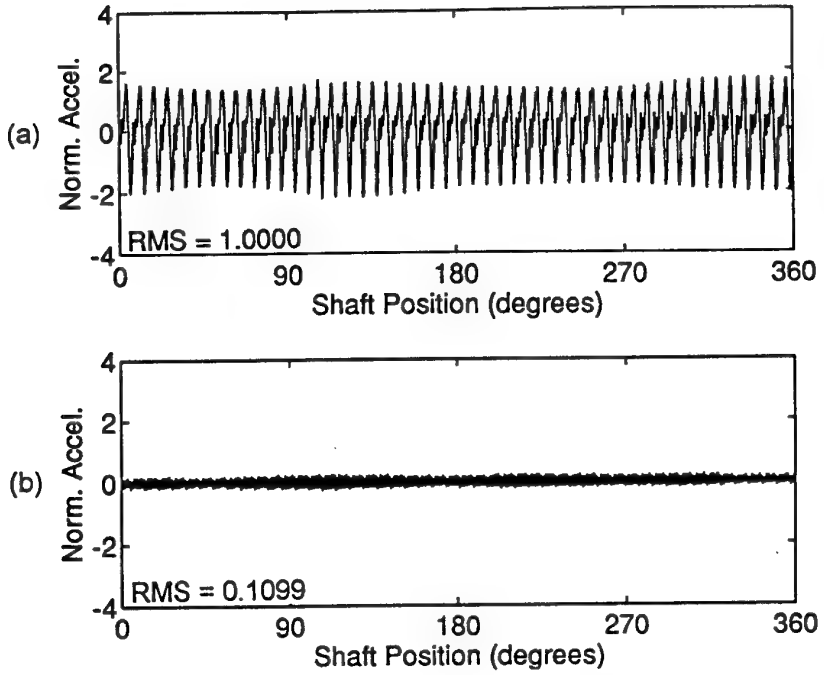


Figure10: Comparison of normalized gear rig output shaft signal averages (No additional filter) a) PC signal average b) GASP difference

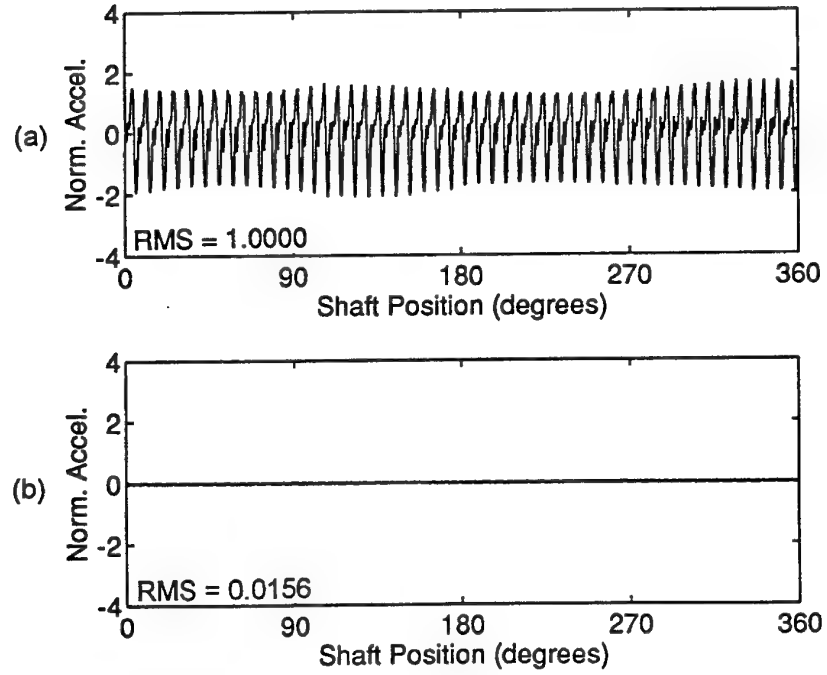


Figure 11: Comparison of normalized gear rig output shaft signal averages (Additional filter @ 270 orders) a) PC Signal average b) GASP difference

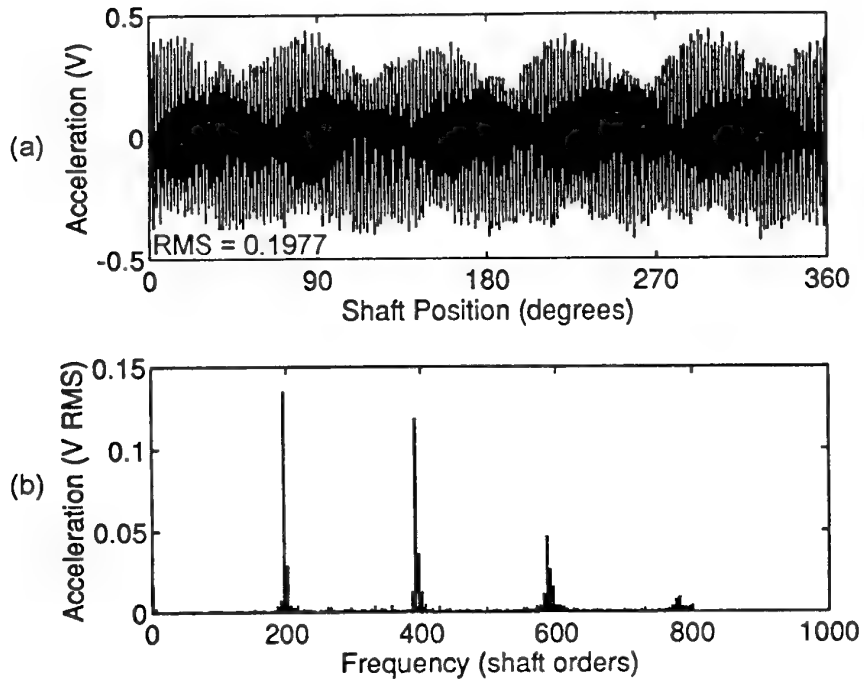


Figure 12: Sea King planet carrier – PC system
a) Signal average b) Spectrum

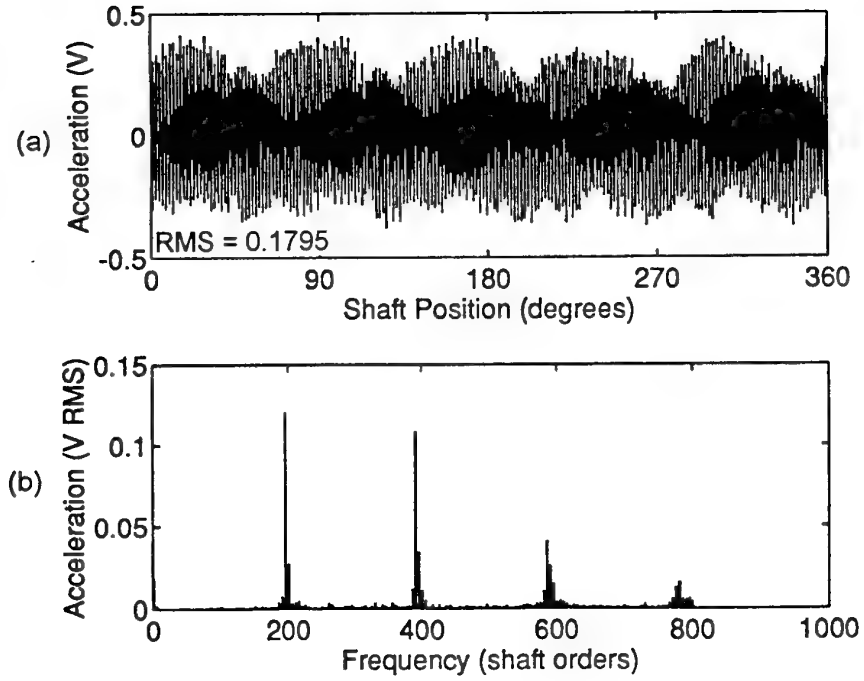


Figure 13: Sea King planet carrier – GASP
a) Signal average b) Spectrum

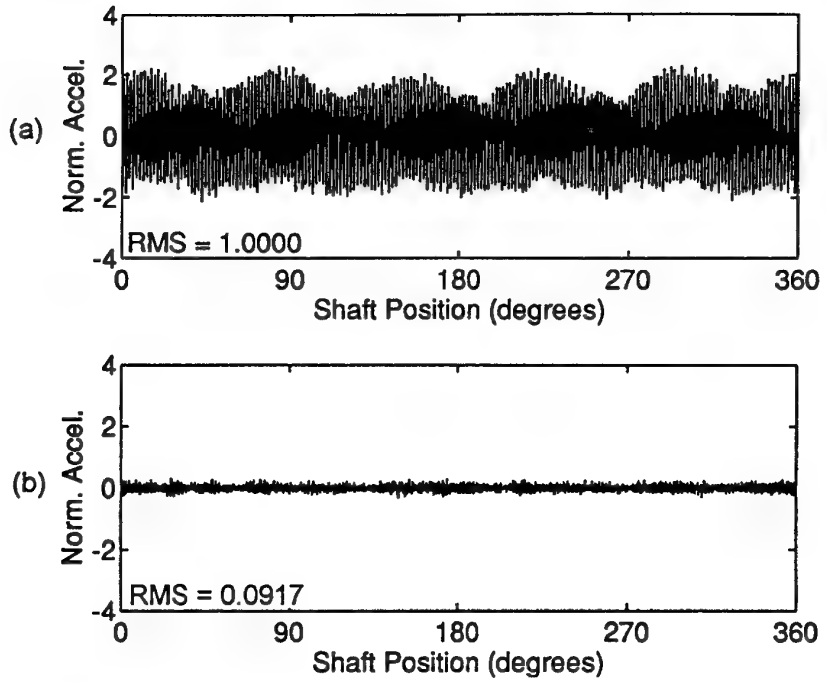


Figure14: Comparison of normalized Sea King planet carrier signal averages (No additional filter) a) PC signal average b) GASP difference

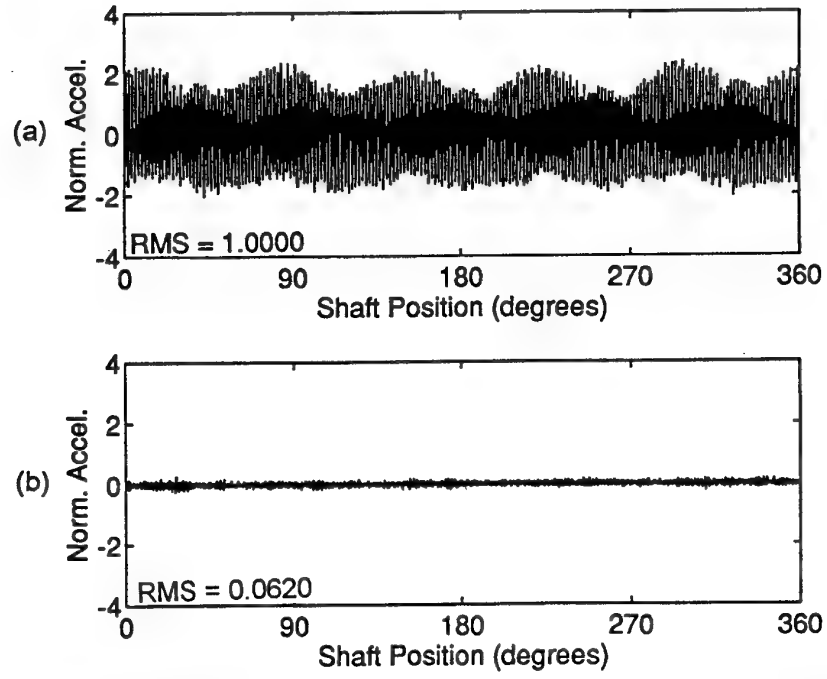


Figure 15: Comparison of normalized Sea King planet carrier signal averages (Additional filter @ 700 orders) a) PC Signal average b) GASP difference

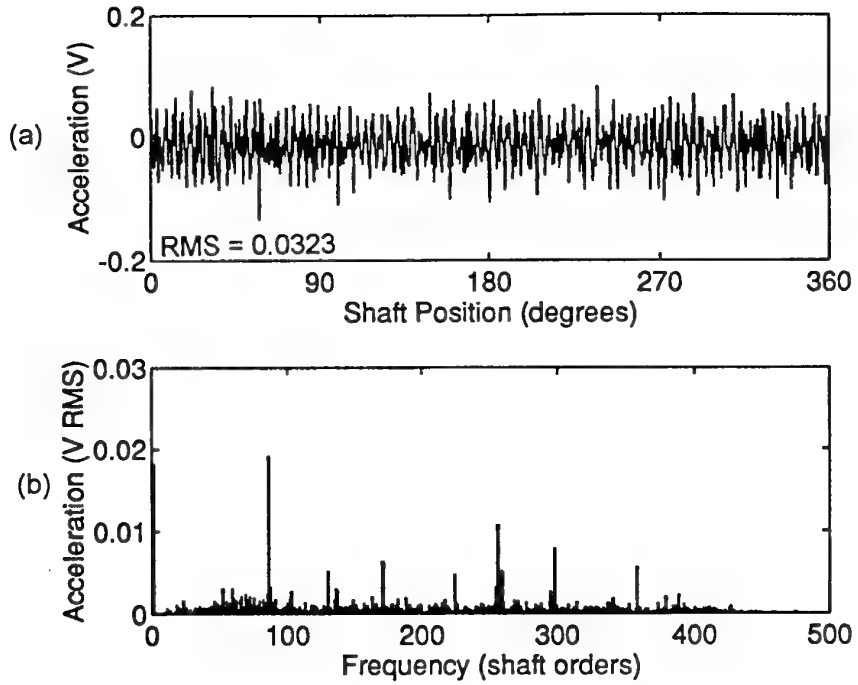


Figure 16: Sea King crown-wheel shaft – PC system
a) Signal average b) Spectrum

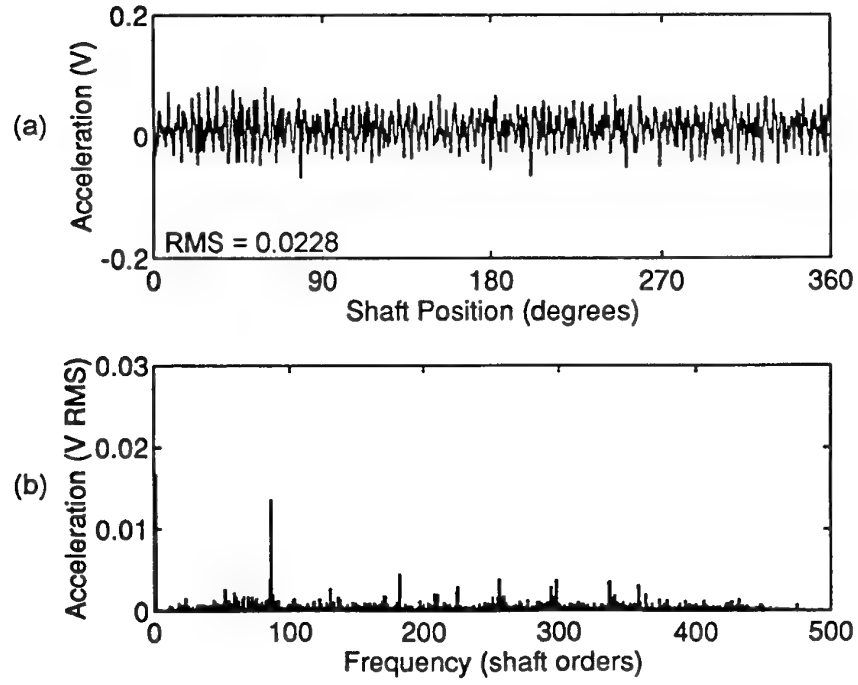


Figure 17: Sea King crown-wheel shaft – GASP
a) Signal average b) Spectrum

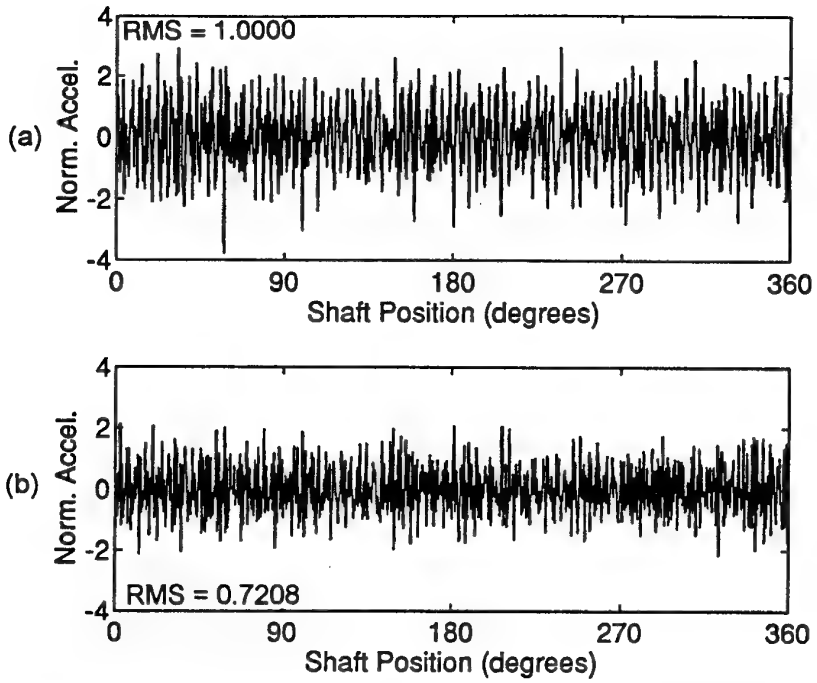


Figure18: Comparison of normalized Sea King crown-wheel shaft signal averages (No additional filter) a) PC signal average b) GASP difference

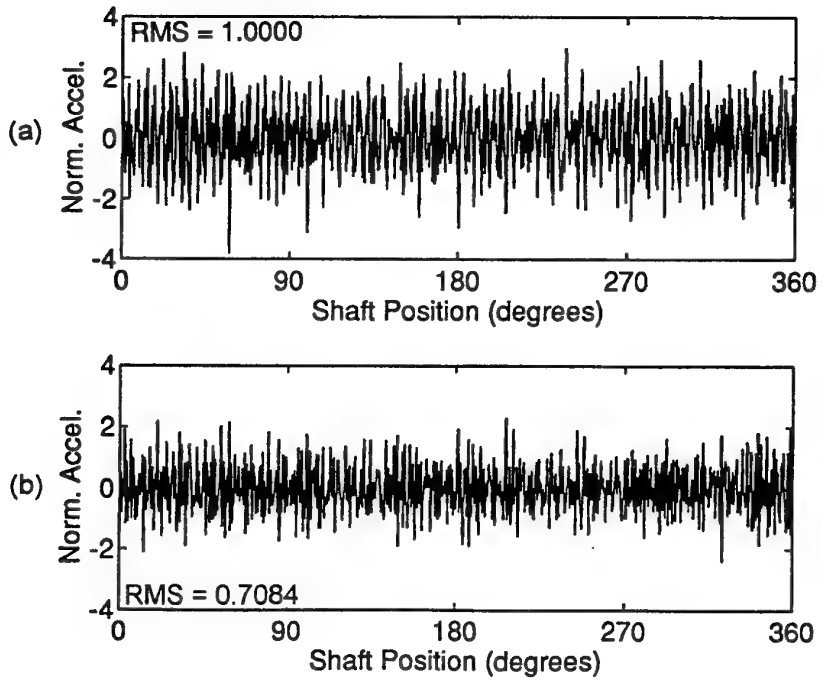


Figure 19: Comparison of normalized Sea King crown-wheel shaft signal averages (Additional filter @ 400 orders) a) PC Signal average b) GASP difference

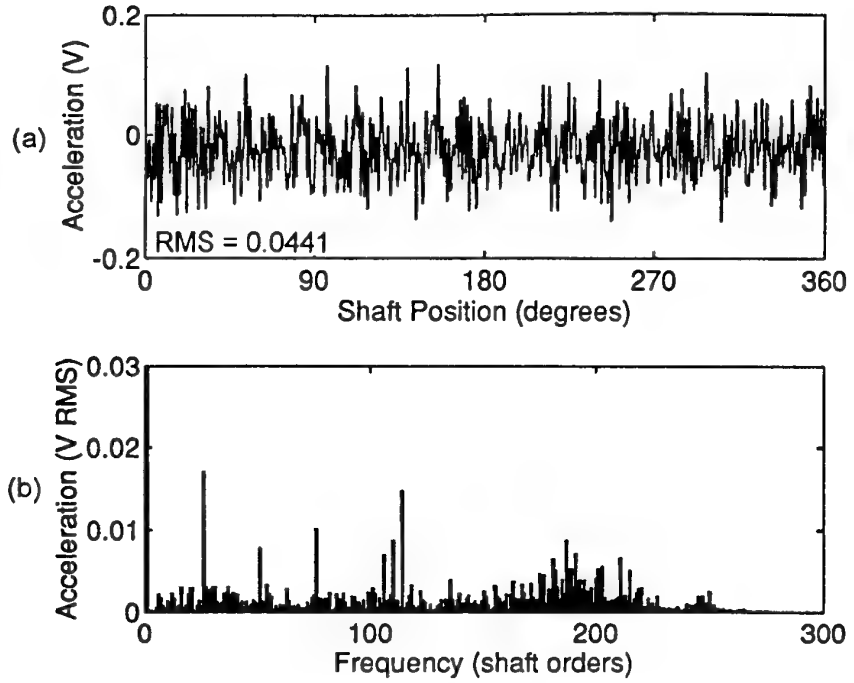


Figure 20: Sea King input pinion shaft - PC system
a) Signal average b) Spectrum

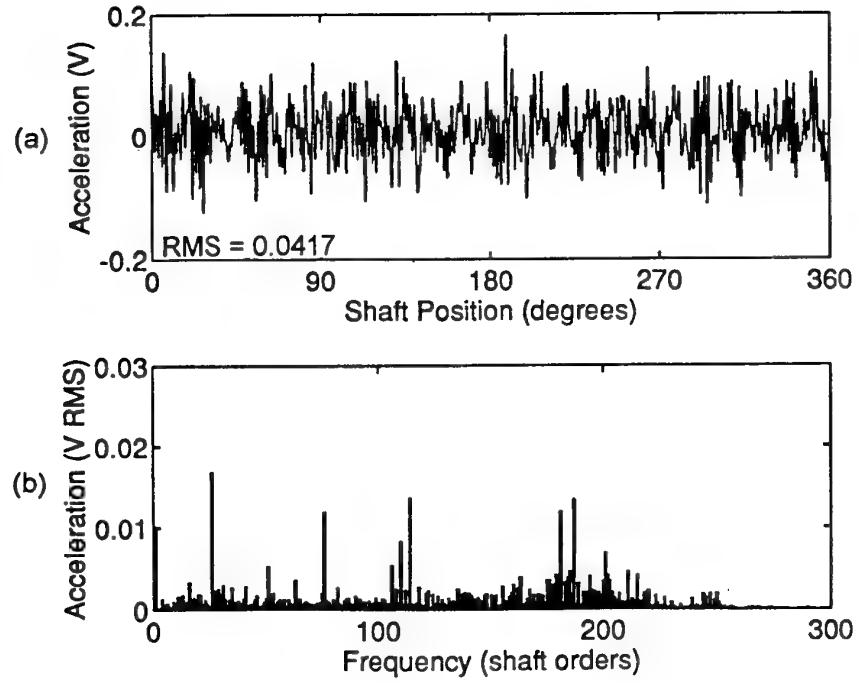


Figure 21: Sea King input pinion shaft - GASP
a) Signal average b) Spectrum

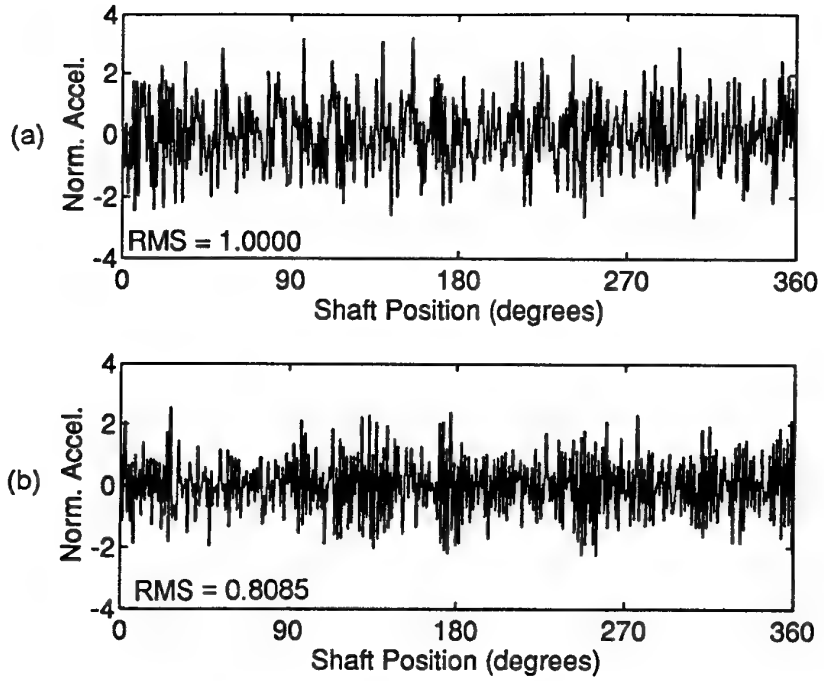


Figure 22: Comparison of normalized Sea King input pinion shaft shaft signal averages (No additional filter) a) PC signal average b) GASP difference

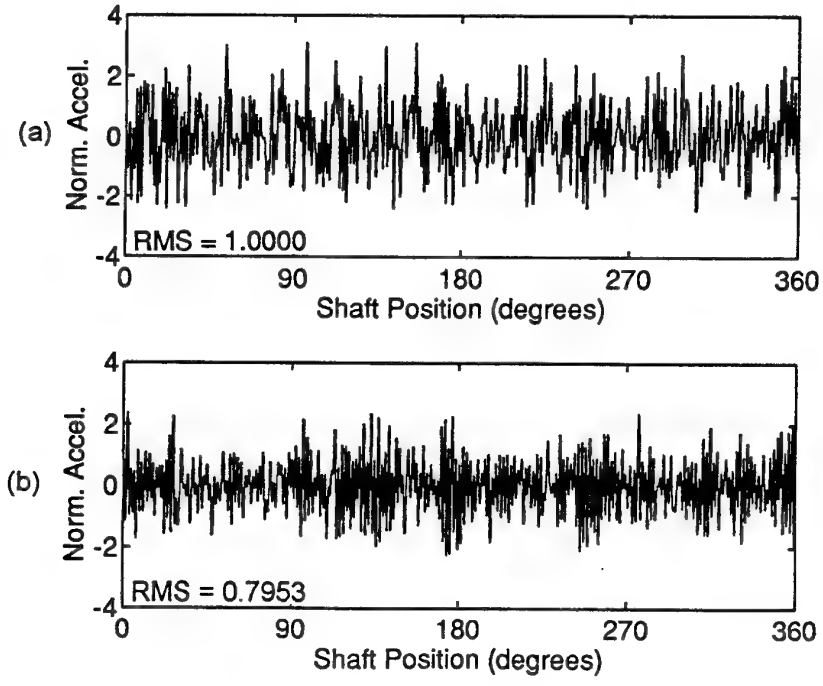


Figure 23: Comparison of normalized Sea King input pinion shaft signal averages (Additional filter @ 235 orders) a) PC Signal average b) GASP difference

Appendix A
COMPSIG Program Code

This program is written in Turbo Pascal 5.0.

```
Program CompSig;
{ D.M. Blunt}
{ This program does the following:
 0. Reads in two real signal averages
 1. Removes the DC offset from each signal average
 2. Low pass filters each signal average
 3. Computes the cross correlation of signal averages
 4. Finds the displacement of the second signal average corresponding to
    the maximum correlation
 5. Removes this phase difference between the signal averages
 6. Normalizes each signal average to have a total energy of one
 7. Saves the modified signal averages in mod1.dat and mod2.dat }

Uses DFT;

{$I ARLVAS.HDR}

TYPE
  extrainfo = record
                PhaseShift,
                FiltCutOff,
                SourceFile1,
                System1,
                DCOffset1,
                ScaleFactor1,
                SourceFile2,
                System2,
                DCOffset2,
                ScaleFactor2    : IDString;
              end;

VAR
  sigavgfile, infofile                                    : file;
  sigavg1, sigavg2, sigim1, sigim2, crossre, crossim,
  real1, real2, imag1, imag2, corrre, corrim, tre, tim : sig_data;
  info1, info2                                            : sig_info;
  maxval, val, fpoint, sfact                              : real;
  mean1, mean2                                            : single;
  i, j, k, maxrot, imin, imax, npoints, np1, npdv2,
  upfreq, ndiv, maxtshift                                 : integer;
  ans                                                     : char;
  addinfo                                                 : extrainfo;
  tstring, infofname                                      : string;


procedure Normalize(var savg: Sig_Data; var scale: real; size: integer);
{ Normalizes the signal average so that the total energy is 1 }
var
  i           : integer;
  totalenergy : real;
begin
  totalenergy := 0.0;
  for i := 0 to size-1 do
    begin
      totalenergy := totalenergy + savg^[i] * savg^[i];
    end;
  scale := 1.0 / sqrt(totalenergy/size);
  for i := 0 to size-1 do  savg^[i] := savg^[i] * scale;
end {Normalize};

procedure shiftavg(var savg: Sig_Data; size, delta: integer);
{ delta = number of places to shift the signal
          - positive shifts to the left
          - negative shifts to the right }
var
  i        : integer;
  tempsig  : Sig_Store;
begin
  tempsig := savg^;
  if delta < 0 then  delta := size + delta;
  for i := 0 to size-1 do  savg^[i] := tempsig[(i+delta) mod size];
end {shiftavg};

procedure timeshift(var specre, specim: sig_data; delpt: real; size: integer);
{ time shift (to the left) the spectrum in specre and specim by the
  fraction of a point in delpt}
var
  deltaphi, phi, cphi, sphi                  : real;
  trecphi,tresphi,timcphi, timsphi           : single;
  n                                          : integer;
begin
  deltaphi := 2.0 * pi * delpt / size;
  for n := 1 to (size div 2 - 1) do
    begin
      phi := n * deltaphi;
      cphi := cos(phi);
      sphi := sin(phi);
      trecphi := specre^[n] * cphi;
      tresphi := specre^[n] * sphi;
```

```
      timcphi := specim^[n] * cphi;
      timsphi := specim^[n] * sphi;
      specre^[n] := trecphi + timsphi;
      specim^[n] := timcphi - tresphi;
      trecphi := specre^[size-n] * cphi;
      tresphi := specre^[size-n] * sphi;
      timcphi := specim^[size-n] * cphi;
      timsphi := specim^[size-n] * sphi;
      specre^[size-n] := trecphi - timsphi;
      specim^[size-n] := timcphi + tresphi;
    end;
end {timeshift};

function sign(x: real): integer;
begin
  if x < 0.0 then sign := -1 else sign := 1
end {sign};


{*************************************************************************}

begin  { main program }
  writeln('***********************************************');
  writeln('Compare Signals');
  writeln;

  if ParamCount < 1 then
  begin
     writeln('USAGE: Compsig SigAvg1 SigAvg2 [ModSigAvg1 ModSigAvg2 InfoFileName]');
     halt(0);
  end;

  assign(sigavgfile,ParamStr(1));
  If not GetSig(sigavgfile,sigavg1,info1) then
     Abort('Illegal or non-existant file '+ParamStr(1));
  if not (info1.Sig_Type in [UnKnown, SigAvg]) then
     Abort('Illegal Signal Type '+ParamStr(1));
  addinfo.sourcefile1 := ParamStr(1);
  if upcase(addinfo.sourcefile1[1]) = 'D' then addinfo.system1 := 'DOLCH'
  else if (upcase(addinfo.sourcefile1[1]) = 'I') or
     (upcase(addinfo.sourcefile1[1]) = 'G') then addinfo.system1 := 'GASP'
  else
    begin
      write(paramstr(1)+' system: ');
      readln(addinfo.system1);
      writeln
    end;

  assign(sigavgfile,ParamStr(2));
  If not GetSig(sigavgfile,sigavg2,info2) then
     Abort('Illegal or non-existant file '+ParamStr(2));
  if not (info2.Sig_Type in [UnKnown, SigAvg]) then
     Abort('Illegal Signal Type '+ParamStr(2));
  addinfo.sourcefile2 := ParamStr(2);
  if upcase(addinfo.sourcefile2[1]) = 'D' then addinfo.system2 := 'DOLCH'
  else if (upcase(addinfo.sourcefile2[1]) = 'I') or
     (upcase(addinfo.sourcefile2[1]) = 'G') then addinfo.system2 := 'GASP'
  else
    begin
      write(paramstr(2)+' system: ');
      readln(addinfo.system2);
      writeln
    end;

  if info1.size <> info2.size then
     Abort('Signal averages are different sizes');
  npoints := info1.size;
  np1 := npoints - 1;
  npdv2 := npoints div 2;

  if (ParamCount > 2) and (ParamCount <= 5) then
    begin
      addinfo.sourcefile1 := ParamStr(3);
      addinfo.sourcefile2 := ParamStr(4);
      infofname := paramstr(5)
    end
  else
    begin
      addinfo.sourcefile1 := 'mod1.dat';
      addinfo.sourcefile2 := 'mod2.dat';
      infofname := 'info.dat'
    end;

  {calc signal statistics}
  mean1 := Mean(sigavg1^[0],npoints);
  mean2 := Mean(sigavg2^[0],npoints);
  write(ParamStr(1):14);
  write(': Mean = ',mean1:8:4);
  LocMinMax(sigavg1^[0],imin,imax,npoints);
  write('  Min = ',sigavg1^[imin]:8:4);
  writeln('  Max = ',sigavg1^[imax]:8:4);
  write(ParamStr(2):14);
  write(': Mean = ',Mean2:8:4);
  LocMinMax(sigavg2^[0],imin,imax,npoints);
  write('  Min = ',sigavg2^[imin]:8:4);
  writeln('  Max = ',sigavg2^[imax]:8:4);
  writeln;
  str(mean1:9:5,addinfo.dcoffset1);
  str(mean2:9:5,addinfo.dcoffset2);
```

```
{subtract means from both signals to remove any dc offset that may be there}
write('Remove means from signals (Y/N): ');
readln(ans);
if (ans = 'y') or (ans = 'Y') then
begin
  for i := 0 to npl do  sigavg1^[i] := sigavg1^[i] - mean1;
  for i := 0 to npl do  sigavg2^[i] := sigavg2^[i] - mean2;
  writeln('The means have been subtracted from the signal averages.');
  addinfo.dcoffset1 := addinfo.dcoffset1 + ' (rem)';
  addinfo.dcoffset2 := addinfo.dcoffset2 + ' (rem)';
end;
writeln;

{allocate memory for working variables
 nb. All these variables are pointers to array[0..4095] of single, but
     they are only allocated memory to cover array[0..npoints-1] of single.
     Therefore to copy one array to another they cannot be assigned to
     each other as in real1^ := real2^ as areas of memory may be
     overwritten.  The elements must be assigned individually.}
Newsig(real1,npoints);
newsig(real2,npoints);
Newsig(imag1,npoints);
newsig(imag2,npoints);
Newsig(sigim1,npoints);
newsig(sigim2,npoints);
newsig(crossre,npoints);
newsig(crossim,npoints);
newsig(corrre,npoints);
newsig(corrim,npoints);
newsig(tre,npoints);
newsig(tim,npoints);

{set imaginary part of signals to zero}
for i := 0 to npl do
  begin
    sigim1^[i] := 0.0;
    sigim2^[i] := 0.0;
  end;

{copy signal averages into working variables}
for i := 0 to npl do
  begin
    real1^[i] := sigavg1^[i];
    real2^[i] := sigavg2^[i];
    imag1^[i] := sigim1^[i];
    imag2^[i] := sigim2^[i];
  end;

{calc fft of each signal}
fft(real1^[0],imag1^[0],npoints,forwrd);
fft(real2^[0],imag2^[0],npoints,forwrd);

{low pass filter each signal if required}
addinfo.filtcutoff := 'none';
Write('Filter signals (Y/N): ');
readln(ans);
if (ans = 'y') or (ans = 'Y') then
begin
  writeln('The signal averages contain freq''s up to ',npdv2-1,' shaft orders');
  upfreq := 0;
  repeat
    write('Upper freq limit in shaft orders (0 = no filtering): ');
    readln(upfreq);
  until (upfreq < npdv2) and (upfreq >= 0);
  if upfreq <> 0 then
    begin
      str(upfreq,addinfo.filtcutoff);
      addinfo.filtcutoff := addinfo.filtcutoff + ' orders';
    end;
  if upfreq > 0 then
    begin
      for i := upfreq to npdv2-1 do
        begin
          real1^[i] := 0.0;
          real1^[npoints-i] := 0.0;
          real2^[i] := 0.0;
          real2^[npoints-i] := 0.0;
          imag1^[i] := 0.0;
          imag1^[npoints-i] := 0.0;
          imag2^[i] := 0.0;
          imag2^[npoints-i] := 0.0;
        end;
      {remove the filtered frequencies from the original signal averages}
      for i := 0 to npl do
        begin
          sigavg1^[i] := real1^[i];
          sigavg2^[i] := real2^[i];
          sigim1^[i] := imag1^[i];
          sigim2^[i] := imag2^[i];
        end;
      fft(sigavg1^[0],sigim1^[0],npoints,inverse);
      fft(sigavg2^[0],sigim2^[0],npoints,inverse);
    end;
end;

{compute cross spectrum}
{nb. Since the signal averages are assumed to be real, the negative
     frequencies of the cross spectrum will be the conjugates of the
     positive frequencies.  This will not be true for the general case
     where the signals may be complex as these will have individual
```

```
     spectra which are not conjugate even about the origin
     (ie F(-f) <> F*(f)). }
for i := 1 to npdv2 - 1 do
  begin
    crossre^[i] := real1^[i] * real2^[i] + imag1^[i] * imag2^[i];
    crossim^[i] := real1^[i] * imag2^[i] - real2^[i] * imag1^[i];
    crossre^[npoints-i] := crossre^[i];
    crossim^[npoints-i] := -crossim^[i];
  end;
crossre^[0] := 0.0;
crossim^[0] := 0.0;
crossre^[npdv2] := 0.0;
crossim^[npdv2] := 0.0;


{find the max value of the cross correlation}
writeln;
repeat
  write('Enter fraction of a point for time shifting (1/x) [1..100]: ');
  readln(ndiv);
until (ndiv >= 1) and (ndiv <= 100);
{copy cross spectrum into working variables}
for i:= 0 to npl do
  begin
    tre^[i] := crossre^[i];
    tim^[i] := crossim^[i];
  end;
for j:= 1 to ndiv do
  begin {time shift loop}
    {copy new cross spectrum into cross corr variables prior to inverse transform}
    for k := 0 to npl do
      begin
        corrre^[k] := tre^[k];
        corrim^[k] := tim^[k];
      end;

    {invert the cross spectrum to get the cross correlation}
    fft(corrre^[0],corrim^[0],npoints,inverse);

    if j = 1 then
      begin
        {save unshifted cross correlation function}
        assign(sigavgfile,'xre.dat');
        rewrite(sigavgfile,1);
        blockwrite(sigavgfile,corrre^,npoints*4);
        close(sigavgfile);
        assign(sigavgfile,'xim.dat');
        rewrite(sigavgfile,1);
        blockwrite(sigavgfile,corrim^,npoints*4);
        close(sigavgfile);

        maxval := corrre^[0];
        maxrot := 0;
        maxtshift := j - 1;
      end;
    for i := 1 to npl do
      begin
        val := corrre^[i];
        if val > maxval then
          begin
            maxval := val;
            maxrot := i;
            maxtshift := j - 1;
          end;
      end;

    {time shift part}
    if (ndiv > 1) and (j < ndiv) then
      begin
        {copy original cross spectrum into working variables}
        for i := 1 to npl do
          begin
            tre^[i] := crossre^[i];
            tim^[i] := crossim^[i];
          end;
        {calc fraction of a point to shift cross corr}
        fpoint := j / ndiv;
        timeshift(tre,tim,fpoint,npoints);
      end;
  end; {time shift loop}
fpoint := maxtshift / ndiv;
writeln;
writeln('Max correlation occurs by shifting ',paramstr(2),' to the left by: ');
write(maxrot,' ',maxtshift,'/',ndiv,' points');
writeln('   (',(maxrot+fpoint) * 360.0 / npoints:1:4,' deg)');
writeln('Maximum correlation is = ',maxval:1:7);

{remove phase difference between signals}
writeln;
write('Shift ',paramstr(2),' by a different amount? (Y/N) ');
readln(ans);
if (ans = 'y') or (ans = 'Y') then
  begin
    repeat
      write('Enter number of points to shift ',paramstr(2),' : ');
      readln(fpoint);
    until (fpoint >= 0.0) and (fpoint < npoints);
    repeat
      write('Enter resolution (1/x points): ');
      readln(ndiv);
```

```
      until (ndiv >= 1) and (ndiv <= 100);
      maxrot := trunc(int(fpoint));
      maxtshift := round(ndiv * frac(fpoint));
      fpoint := maxtshift / ndiv;
    end;
  timeshift(real2,imag2,fpoint,npoints);
  str(maxrot,tstring);
  addinfo.phaseshift := tstring;
  if ndiv <> 1 then
    begin
      str(maxtshift,tstring);
      addinfo.phaseshift := addinfo.phaseshift + ' ' + tstring + '/';
      str(ndiv,tstring);
      addinfo.phaseshift := addinfo.phaseshift + tstring;
    end;
  addinfo.phaseshift := addinfo.phaseshift + ' points';
  for i := 0 to npl do
    begin
      sigavg2^[i] := real2^[i];
      sigim2^[i] := imag2^[i];
    end;
  fft(sigavg2^[0],sigim2^[0],npoints,inverse);
  shiftavg(sigavg2,npoints,maxrot);
  {no need to shift imaginary part as it should be zero for a real signal}

  {for test purposes save the imaginary parts of the modified signal averages
   to see whether they are non zero which would indicate an error}
  assign(sigavgfile,'sim1.dat');
  rewrite(sigavgfile,1);
  blockwrite(sigavgfile,sigim1^,npoints*4);
  close(sigavgfile);
  assign(sigavgfile,'sim2.dat');
  rewrite(sigavgfile,1);
  blockwrite(sigavgfile,sigim2^,npoints*4);
  close(sigavgfile);

  {normalize each signal}
  Normalize(sigavg1,sfact,npoints);
  str(sfact:9:5,addinfo.scalefactor1);
  Normalize(sigavg2,sfact,npoints);
  str(sfact:9:5,addinfo.scalefactor2);

  {save modified signals}
  assign(sigavgfile,addinfo.sourcefile1);
  PutSig(sigavgfile,sigavg1,info1);

  assign(sigavgfile,addinfo.sourcefile2);
  PutSig(sigavgfile,sigavg2,info2);

  assign(infofile,infofname);
  rewrite(infofile,1);
  blockwrite(infofile,addinfo,sizeof(extrainfo));
  close(infofile);

  FreeSig(sigavg1,npoints);
  FreeSig(sigavg2,npoints);
  FreeSig(sigim1,npoints);
  FreeSig(sigim2,npoints);
  FreeSig(real1,npoints);
  FreeSig(real2,npoints);
  FreeSig(imag1,npoints);
  FreeSig(imag2,npoints);
  FreeSig(crossre,npoints);
  FreeSig(crossim,npoints);
  FreeSig(corrre,npoints);
  FreeSig(corrim,npoints);
  freesig(tre,npoints);
  freesig(tim,npoints);
end.
```

Appendix B
Anti-Aliasing Filter Characteristics

The filter characteristics for the anti-aliasing filters are shown in Figures B1 to B5. Both filter corner frequencies were set to 15 kHz.

Fig B1. GASP Filter Characteristic - Magnitude (15 kHz Cut-off)

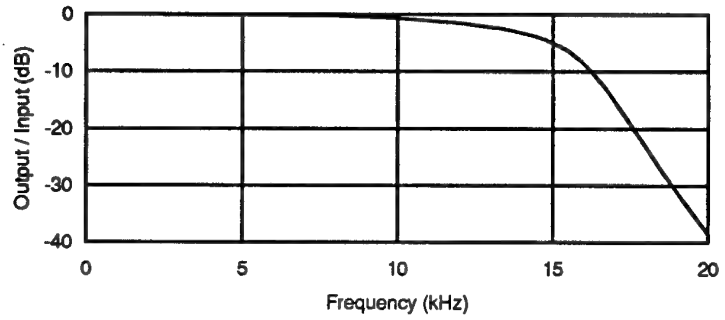


Fig B2. GASP Filter Characteristic - Phase (15 kHz Cut-off)

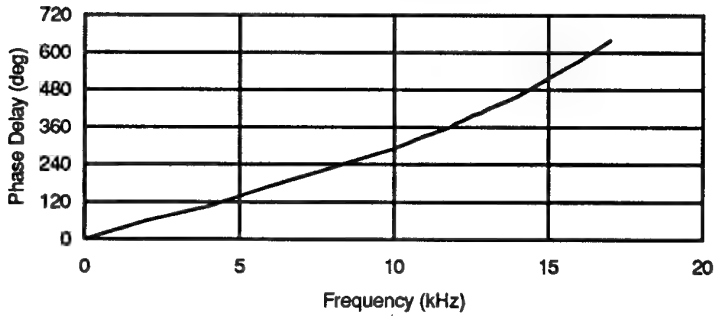


Fig B3. GASP Filter Cut-off Frequency

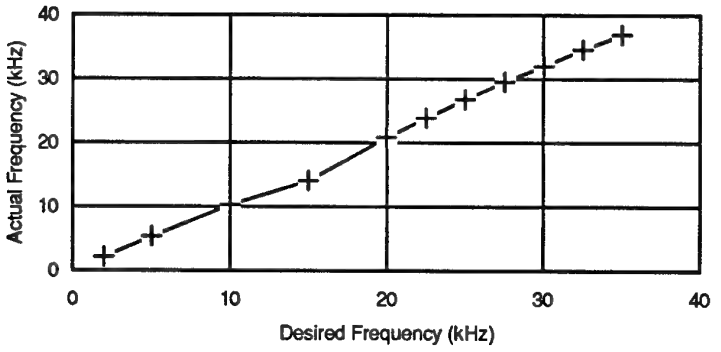


Fig B4. TechFilter Characteristic - Magnitude (15 kHz Cut-off)

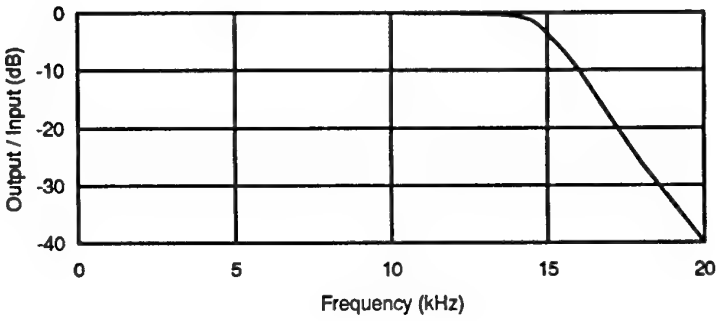
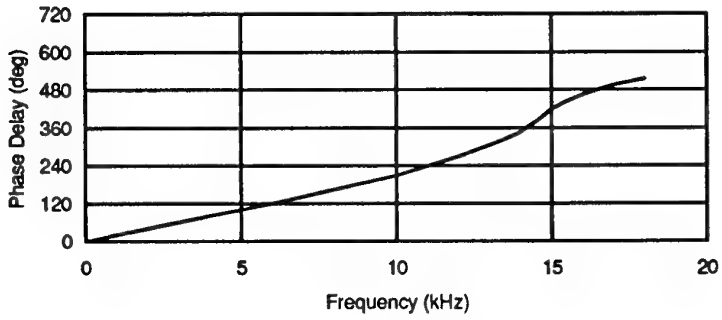


Fig B5. TechFilter Characteristic - Phase (15 kHz Cut-off)



Appendix C
GASP Signal Averaging Programs

These programs were written by O.F. Holland, Instrumentation and Trials Group, AMRL.

```
/*
 *
 *  D S P A V R G
 *
 * produce average of gear vibration for Gearbox Signal from DSP32C memory
 *
 * dspAvrg.c  last edited Mon Jul 06 16:03:49 1992 by O.F.H.
 *
 * 0v0, Jun, 92, O.F.H., original code from dspCGASP.c 1v1 & convTach.c 0v0
 *
 *
 * RUN thus ;-
 *  d3sim -T -C test.cfg -c -e dspAvrg
 *
 *
 * CREATE with AT & T C compiler thus ;-
 *  d3as -Q -l minStart.s
 *  d3cc -l -c dspAvrg.c
 *  d3cc -m dsmem32c.map -s minStart.o -o dspAvrg dspAvrg.o -lap
 *
 * RESULT should be a zero written to the parallel port and data in memory
 *
 * Written by O.F.Holland, Jun 92.
 *
 */

#define  FALSE ((int)0)
#define  TRUE (!FALSE)


#include <libap.h>
#include <math.h> /* floor() */

/*
  struct passedInfo
  {
        long  countsBetweenSamples;
        long  maxNumberOfSamples;
        long  startOfADCData;
        long  startOfTachoData;
        long  endOfADCData;
        long  endOfTachoData;
        long  minTachoPeriod;
        long  startOfAverageData;
        long  averagesRequired;
        long  pointsAroundGOIrequired;
        float  gearRatioConstant;
        float  scaleFactorConstant;
  };
*/
int  countsBetweenSamples;
int  numberOfSamples;
int  *startOfADCData;
int  *startOfTachoData;
int  *endOfADCData;
int  *endOfTachoData;
int  minPeriod;
float  *startOfAverageBuffer;
int  averagesRequired;
int  pointsAroundGOIrequired;
float  gearRatio;
float  scaleFactor;

int  *periodPtr;
int  offset;
int  intCompleteRevs;
float  invPointsRequiredAroundGOI;
float  invCountsBetweenSamples;
float  wordReferenceAddress;


extern float  doCubicInterpolation();

void  initGlobalVariables();
void  initDataBuffer();
void  doAverage();
void  scaleResults();
float  calcPointPositionInTermsOfGOIrevs();


main()
{
    dsp32( ( short )2, &gearRatio );
 /* Clear the Averaged Data buffer space */
    initDataBuffer();
    initGlobalVariables();
```

```
    doAverage();
    scaleResults();
    ieee32( ( short ) pointsAroundGOIrequired, startOfAverageBuffer );
    return( 0 );
}

void  initDataBuffer()
{
    int  cnt;
    float  *ptr;

    ptr = startOfAverageBuffer;
    for( cnt = pointsAroundGOIrequired; cnt-- > 0; *ptr++ = 0.0 );
}

void  scaleResults()
{
    int  cnt;
    float  *ptr;

    ptr = startOfAverageBuffer;
    for( cnt = pointsAroundGOIrequired; cnt-- > 0; *ptr++ *= scaleFactor );
}

void  initGlobalVariables()
{
    invPointsRequiredAroundGOI = inv(( float ) pointsAroundGOIrequired );
    invCountsBetweenSamples = inv(( float ) countsBetweenSamples );
    scaleFactor = scaleFactor * inv(( float ) averagesRequired );
    intCompleteRevs = 0;
    periodPtr = startOfTachoData;
    offset = *periodPtr--;
    offset = offset - *periodPtr--;
    wordReferenceAddress = 5.0 + (( float )( *periodPtr - offset )) *
        invCountsBetweenSamples;
    offset -= ( *periodPtr-- % countsBetweenSamples );
    if( offset < 0 )  offset += countsBetweenSamples;
}
```

```
/*
 *
 *  D O A V E R A G
 *
 * optimized averaging process for dspAvrg.c
 *
 * doAverag.s  last edited Wed Jul 08 10:24:44 1992 by O.F.H.
 *
 * 0v1, Jul, 92, O.F.H., replaced modf( gearRatio ... ) with duac ... etc
 * 0v0, Jul, 92, O.F.H., original code from dspAvrg.c 0v0
 *
 *
 *
 * CREATE with AT & T C compiler thus ;-
 *  d3as -Q -l doaverag.s
 *  d3cc -m dsmem32c.map -s minStart.o -o dspAvrg dspAvrg.o intrplt.o doaverag.o -lap -lm
 *
 * Written by O.F.Holland, Jul 92.
 *
 */


/*
void  doAverage()
{
    register int  revsCompleted;
    register int  point;
    register float  *averagePtr;
    float  completeRevs, fractionOfRev;

    for( revsCompleted = 0; revsCompleted < averagesRequired; revsCompleted++ )
    {
        averagePtr = startOfAverageBuffer;
        for( point = 0; point < pointsAroundGOIrequired; point++ )
        {
            fractionOfRev = modf( gearRatio * (( float ) revsCompleted +
                ( float ) point * invPointsRequiredAroundGOI ),  &completeRevs );
            if( intCompleteRevs < ( int ) completeRevs )
            {
                wordReferenceAddress = floor( wordReferenceAddress ) + 1.0;
                wordReferenceAddress += (( float )( *periodPtr - offset )) *
invCountsBetweenSamples;
                offset -= *periodPtr-- % countsBetweenSamples;
                if( offset < 0 )  offset += countsBetweenSamples;
                intCompleteRevs++;
            }
            *averagePtr++ += doCubicInterpolation( wordReferenceAddress +
                invCountsBetweenSamples * fractionOfRev * ( float ) *periodPtr );
        }
    }
}
*/


#define  PERIOD_PTR  r10
#define  PERIOD_PTRe  r10e
#define  AVERAGE_PTR  r11
#define  AVERAGE_PTRe  r11e
#define  POINT  r12
#define  POINTe  r12e
#define  REVS_COMPLETED  r13
#define  REVS_COMPLETEDe  r13e
#define  FRACTION_OF_REV  r14 - 24
#define  COMPLETE_REVS  r14 - 28
#define  SP_4_EXIT  r14 - 28
#define  SP_4_RESTORE  r14 - 20


        .rsect  ".data"
        .align  4
flt1p0:
        float   1.00000000e+000
        .rsect  ".text"
        .global doAverage
doAverage:
        r14e = r14 + 8
        *r14++ = r18e
        *r14++ = REVS_COMPLETEDe
        *r14++ = POINTe
        *r14++ = AVERAGE_PTRe
        *r14++ = PERIOD_PTRe
        r1e = periodPtr
        PERIOD_PTRe = *r1
/*  for( revsCompleted = 0; .... */
        goto L162
        REVS_COMPLETEDe = -1
L163:
/*  averagePtr = startOfAverageBuffer; */
        r1e = startOfAverageBuffer
        AVERAGE_PTRe = *r1
/*  for( point = 0;  .... */
        goto L166
        POINTe = -1
L167:
/*  fractionOfRev = modf( gearRatio * (( float ) revsCompleted +
        ( float ) point * invPointsRequiredAroundGOI ),  &completeRevs ); */
        *r14 = POINTe
        a1 = float24( *r14 )
        r2e = COMPLETE_REVS
```

```
        *r14 = REVS_COMPLETEDe
        a0 = float24( *r14 )
        r1e = invPointsRequiredAroundGOI
        a1 = a1 * *r1
        r1e = gearRatio
        a0 = *r1 * a0
        *r14 = a0 = a0 + *r1 * a1
        nop
        dauc = 0x1c
        *r2 = a0 = int24(a0)
        nop
        r1e = FRACTION_OF_REV
        a0 = float24(a0)
        *r1 = a0 = -a0 + *r14
        dauc = 0xc
/*  if( intCompleteRevs < ( int ) completeRevs ) */
        r1e = intCompleteRevs
        r1e = *r1
        r2e = *r2
        nop
        r1e - r2
        if (ge) goto L169
/*  wordReferenceAddress = floor( wordReferenceAddress ) + 1.0; */
        r1e = wordReferenceAddress
        dauc = 0x1c
        a0 = int24(*r1)
        nop
        r1e = flt1p0
        a0 = float24(a0)
        dauc = 0xc
        a0 = a0 + *r1
/*  wordReferenceAddress += (( float )( *periodPtr - offset )) * invCountsBetweenSamples;
*/
        r1e = *PERIOD_PTR
        r2e = offset
        r2e = *r2
        nop
        r1e = r1 - r2
        *r14 = r1e
        a1 = float24( *r14 )
        r1e = invCountsBetweenSamples
        r3e = wordReferenceAddress
        *r3 = a0 = a0 + a1 * *r1
/*  offset -= *periodPtr-- % countsBetweenSamples; */
        r2e = countsBetweenSamples
        r2e = *r2
        r3e = *PERIOD_PTR--
        *r14++r19 = r2e
        *r14++r19 = r3e
        call _s24mod (r18)
L172:
        r18e = L172+4
        r14e = r14 - 8
        r3e = offset
        r2e = *r3
        nop
        r1e = r2 - r1
        *r3 = r1e
        nop
/*  if( offset < 0 )  offset += countsBetweenSamples; */
        r1e = *r3
        r2e = countsBetweenSamples
        if (ge) goto L173
        r4e = intCompleteRevs
        r2e = *r2
        nop
        r1e = r1 + r2
        *r3 = r1e
/*  intCompleteRevs++; */
L173:
        r2e = *r4
        nop
        r2e = r2 + 1
        *r4 = r2e
/*  *averagePtr++ += doCubicInterpolation( wordReferenceAddress +
        invCountsBetweenSamples * fractionOfRev * ( float ) *periodPtr ); */
L169:
        r2e = invCountsBetweenSamples
        r3e = FRACTION_OF_REV
        a0 = *r2 * *r3
        a1 = float24( *PERIOD_PTR )
        nop
        r1e = wordReferenceAddress
        *r14++ = a0 = *r1 + a0 * a1
        nop
        call doCubicInterpolation (r18)
L174:
        r18e = L174+4
        r14e = r14 - 4
        *AVERAGE_PTR++ = a0 = *AVERAGE_PTR + a0
L166:
        r1e = pointsAroundGOIrequired
        r1e = *r1
        POINTe = POINT + 1
        POINTe - r1
        if (lt) goto L167
```

```
L162:
        r1e = averagesRequired
        r1e = *r1
        REVS_COMPLETEDe = REVS_COMPLETED + 1
        REVS_COMPLETEDe - r1
        if (lt) goto L163
        r1e = periodPtr
/*  restore periodPtr value before exit */
        *r1 = PERIOD_PTRe
        r3e = SP_4_RESTORE
        r18e = *r3++
        REVS_COMPLETEDe = *r3++
        POINTe = *r3++
        AVERAGE_PTRe = *r3++
        PERIOD_PTRe = *r3++
        return (r18)
        r14e = SP_4_EXIT
        .rsect  ".data"
```

```
/*
 *
 *  I N T R P L T
 *
 * DSP32C machine code version of the Cubic Interpolation
 *
 * intrplt.s  last edited Wed Jul 01 17:47:30 1992 by O.F.H.
 *
 * 0v1, Jul, 92, O.F.H., optimized code to remove r8 & r9 useage
 * 0v0, Jun, 92, O.F.H., original code from dspAvrg.c 0v0
 *
 *
 * CREATE with AT & T C compiler thus ;-
 *  d3as -Q -l intrplt.s
 *
 * USES registers  a0, a1, r1, r2, & r3 ( destroyed ).
 *      registers  a3 ( saved & restored on exit ).
 *
 * Written by O.F.Holland, Jun 92.
 *
 */

        .rsect  ".data"
        .align  4
FLT2p0: float   2.00000000e+000
FLT0p5: float   5.00000000e-001
FLT0p3r:
        float   3.33333333e-001

        .rsect  ".text"
        .global doCubicInterpolation
doCubicInterpolation:
        *r14++ = a3 = a3
    /*  Convert float on stack to integer ( stacked ) and remainder ( a3 )*/
        dauc = 0x1c
        r1e = r14 - 8
        *r14++ = a0 = int24(*r1)
        r3e = r1 + 12
        r2e = r1 + 8
        a0 = float24(a0)
        a3 = -a0 + *r1
    /*  Convert stacked integer to register word address */
        r2e = *r2
        dauc = 0xc
    /*  Convert integer to byte address */
        r2e = r2 * 2
    /*  Convert integer to pointer to start of 4 integer array */
        r1e = r2 - 2
    /*  Convert short integers in array to floats on the stack */
        *r14++ = a0 = float( *r1++ )
        *r14++ = a0 = float( *r1++ )
        *r14++ = a0 = float( *r1++ )
        *r14++ = a0 = float( *r1++ )
        r2e = r3 + 4
/*  y11 = *y++ = *y1 + *y1 * xVal - *y0 * xVal; */
        a0 = *r2++ + a3 * *r2
        *r14++ = a0 = a0 - a3 * *r3++
/*  y21 = *y++ = *y0 - *y0 * xVal + *y1 * xVal; */
        a0 = *r3++ - a3 * *r3
        *r14++ = a1 = a0 + a3 * *r2++
/*  y31 = *y++ = 2.0 * *y0 + *y1 * xVal - *y0 * xVal - *y1; */
        r1e = FLT2p0
        a0 = - *r2++ + a3 * *r2
        a0 = a0 + *r1 * *r3
        *r14++ = a0 = a0 - a3 * *r3
/*  y12 = *y++ = 0.5 * ( *y2 + *y3 + *y3 * xVal - *y2 * xVal ); */
        r3e = r2 + 4
        a0 = *r3++ + a3 * *r3
        a1 = *r2++ - a3 * *r2
        r1e = FLT0p5
        a0 = *r1 * a0
        a0 = a0 + *r1 *a1
/*  y22 = *y++ = *y2 + 0.5 * ( *y3 * xVal - *y2 * xVal ); */
        a1 = *r3-- * a3
        a1 = a1 - a3 * *r2
        nop
        r2e = FLT2p0
        a1 = *r3 + a1 * *r1
/* result = *y = 0.333333333 * ( 2.0 * *y2 - *y2 * xVal + *y3 * xVal + *y3 ); */
        r3e = r14 - 36
        a0 = a1 - a3 * a0
        a0 = a0 + a3 * a1
        a0 = a0 + *r2 * a0
        a3 = *r3
        r1e = FLT0p3r
        a0 = *r1 * a0
        return  (r18)
        r14e = r14 - 36

.rsect  ".data"
```

DISTRIBUTION

AUSTRALIA

DEFENCE ORGANISATION

Defence Science and Technology Organisation

Chief Defence Scientist
FAS Science Policy
AS Science Corporate Management
} shared copy

Counsellor Defence Science, London (Doc Data Sheet only)
Counsellor Defence Science, Washington (Doc Data Sheet only)
Senior Defence Scientific Adviser (Doc Data Sheet only)
Scientific Advisor Policy and Command (Doc Data Sheet only)
Navy Scientific Adviser (3 copies Doc Data Sheet only)
Scientific Adviser - Army (Doc Data Sheet only)
Air Force Scientific Adviser

Aeronautical and Maritime Research Laboratory
Director
Library Fishermens Bend
Library Maribyrnong
Chief Airframes and Engines Division
Author: D.M. Blunt
B. Rebbechi
B.D. Forrester
M. Shilo
C.L. Dolan
J.F. Harvey
O.F. Holland
I.M. Kerton

Electronics and Surveillance Research Laboratory
Director
Main Library - DSTO Salisbury

Defence Central

OIC TRS, Defence Central Library
Document Exchange Centre, DSTIC (8 copies)
Defence Intelligence Organisation
Library, Defence Signals Directorate (Doc Data Sheet Only)

Air Force

DTA-LC
OIC ATF, ATS, RAAFSTT, WAGGA (2 copies)

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering

OTHER GOVERNMENT DEPARTMENTS AND AGENCIES

AGPS

OTHER ORGANISATIONS

NASA (Canberra)

SPARES (7 COPIES)

TOTAL (40 COPIES)

AL 149 DEPARTMENT OF DEFENCE

DOCUMENT CONTROL DATA

PAGE CLASSIFICATION
UNCLASSIFIED

PRIVACY MARKING

1a. AR NUMBER	1b. ESTABLISHMENT NUMBER	2. DOCUMENT DATE	3. TASK NUMBER
AR-008-933	DSTO-TR-0067	SEPTEMBER 1994	91/063

4. TITLE

AN EVALUATION OF THE GEAR AVERAGING SIGNAL PROCESSOR (GASP)

5. SECURITY CLASSIFICATION (PLACE APPROPRIATE CLASSIFICATION IN BOX(S) IE. SECRET (S), CONF. (C) RESTRICTED (R), LIMITED (L), UNCLASSIFIED (U)).

DOCUMENT	TITLE	ABSTRACT
U	U	U

6. NO. PAGES
44

7. NO. REFS.
5

8. AUTHOR(S)

D.M. Blunt

9. DOWNGRADING/DELIMITING INSTRUCTIONS

Not applicable.

10. CORPORATE AUTHOR AND ADDRESS

AERONAUTICAL AND MARITIME RESEARCH LABORATORY
AIRFRAMES AND ENGINES DIVISION
GPO BOX 4331
MELBOURNE VIC 3001 AUSTRALIA

11. OFFICE/POSITION RESPONSIBLE FOR:

RAAF DTA-LC

SPONSOR -

SECURITY -

DOWNGRADING CAED

APPROVAL

12. SECONDARY DISTRIBUTION (OF THIS DOCUMENT)

Approved for public release.

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DSTIC, ADMINISTRATIVE SERVICES BRANCH, DEPARTMENT OF DEFENCE, ANZAC PARK WEST OFFICES, ACT 2601

13a. THIS DOCUMENT MAY BE ANNOUNCED IN CATALOGUES AND AWARENESS SERVICES AVAILABLE TO ....

No limitations.

14. DESCRIPTORS

Gear boxes
Shafts (Machine elements)
Vibration analysis
Signal processing

15. DISCAT SUBJECT CATEGORIES

1309

16. *ABSTRACT*

The Gear Averaging Signal Processor (GASP) is an IBM PC XT board designed and built by AMRL to calculate synchronous vibration signal averages in real-time. This report describes GASP at an initial stage of development in which it produces signal averages in a non-real-time mode. To evaluate GASP in this mode of operation, a comparison was made with a PC-based signal averaging system. This comparison shows that GASP works effectively, but is limited by the memory available on the board and the precision of some of its floating point calculations.

PAGE CLASSIFICATION
UNCLASSIFIED
PRIVACY MARKING

THIS PAGE IS TO BE USED TO RECORD INFORMATION WHICH IS REQUIRED BY THE ESTABLISHMENT FOR ITS OWN USE BUT WHICH WILL NOT BE ADDED TO THE DISTIS DATA UNLESS SPECIFICALLY REQUESTED.

16. ABSTRACT (CONT).

17. IMPRINT

AERONAUTICAL AND MARITIME RESEARCH LABORATORY, MELBOURNE

18. DOCUMENT SERIES AND NUMBER	19. WA NUMBER	20. TYPE OF REPORT AND PERIOD COVERED
Technical Report TR-0067	43 421F	

21. COMPUTER PROGRAMS USED

22. ESTABLISHMENT FILE REF.(S)

M1/9/28

23. ADDITIONAL INFORMATION (AS REQUIRED)